Werkhefte der
Universität Tübingen
Reihe A:
Naturwissenschaften

Rudolf Schief

Einführung in die Mikroprozessoren und Mikrocomputer

Am Beispiel der Mikroprozessoren

8080

8085

Z80

ATTEMPTO
Verlag Tübingen
GmbH

Rudolf Schief +++ Einführung in die Mikroprozessoren und Mikrocomputer

Rudolf Schief

E I N F Ü H R U N G

I N D I E

M I K R O P R O Z E S S O R E N

U N D

M I K R O C O M P U T E R

A m B e i s p i e l d e r M i k r o p r o z e s s o r e n

8 0 8 0
8 0 8 5
Z 8 0

Sechste, überarbeitete Auflage

ATTEMPTO Verlag
Tübingen
GmbH

Dr.-Ing. Rudolf Schief

Professor für Elektronik
an der Fakultät für Physik
der Universität Tübingen

6. überarbeitete Auflage

Alle Rechte vorbehalten
© 1983 ATTEMPTO Verlag GmbH
Tuebingen University Press

Gesamtherstellung Druckerei Tübinger Chronik
Printed in Germany
ISBN 3 - 921552 - 41 - 9

Vorwort zur ersten Auflage

Das vorliegende Buch entstand aus einer Vorlesung, die ich im Wintersemester 1978/79 vor Studenten der Physik an der Universität Tübingen als "Einführung in die programmierte Elektronik" gehalten habe. Das Buch ist bewußt im Stil einer Vorlesung gehalten. Es soll - ohne die letzte äußere Perfektion - möglichst rasch den Studenten an die Hand gegeben werden können, damit sie den Einstieg in dieses aktuelle Gebiet der Elektronik nicht verpassen und den Anschluß an eine Entwicklung behalten, die alle Gebiete der Naturwissenschaften und Technik nachhaltig beeinflussen wird.

Dem Präsidenten der Eberhard-Karls-Universität Tübingen, Herrn A. THEIS, bin ich zu Dank verpflichtet, daß er mich dazu überredet hat, die Mühen der Ausarbeitung einer Vorlesung auf mich zu nehmen. Insbesondere danke ich ihm dafür, daß dieses Buch im ATTEMPTO-Verlag der Universität Tübingen so schnell fertiggestellt werden konnte.
Der Firma SIEMENS danke ich für die großzügige Bereitstellung von ausführlichem Informationsmaterial über den Mikroprozessor SAB 8080 A und ihre stets wohlwollende Unterstützung.

Tübingen, im Juni 1979

Vorwort zur zweiten Auflage

Anläßlich der sehr schnell notwendig gewordenen 2. Auflage wurde der Stoff durchgesehen und auf den aktuellen Stand gebracht. Als Erweiterung wurde der Mikroprozessor 8085 mit in das Buch aufgenommen.

Tübingen, im Januar 1980

Vorwort zur fünften Auflage

Bei der 5. Auflage wurde der Stoff überarbeitet und auch der Mikroprozessor Z80 als "naher Verwandter des 8080" mit in das Buch aufgenommen. Den Firmen ZILOG und MOSTEK danke ich herzlich für die Bereitstellung von ausführlichem Informationsmaterial.

Tübingen, im Dezember 1982

Vorwort zur sechsten Auflage

Anläßlich der rasch notwendig gewordenen 6. Auflage wurde der Stoff überarbeitet und auf den aktuellen Stand gebracht.

Tübingen, im September 1983

Rudolf Schief

Inhaltsverzeichnis

Einleitung 1

1) Darstellung von Information durch Bitmuster 7
   1.1) Die Begriffe Bit, Byte, Baud 7
   1.2) Darstellung von Zahlen durch Bitmuster 8
   1.3) Darstellung von Zeichen durch Bitmuster 13
   1.4) Darstellung von Maschinenbefehlen durch Bitmuster 15
   1.5) Beispiel 17

2) Einführung in die Mikrocomputer 18
   2.1) Überblick über ein Mikrocomputersystem 18
   2.2) Ausführungsformen von Mikroprozessoren 21
   2.3) Mikroprozessortypen und Hersteller 23
   2.4) Zum Mikroprozessor 8080 25

3) Grobstruktur des Mikrocomputers 26
   3.1) Bus-Struktur 27
   3.2) Zentraleinheit 28
   3.3) Speicher 31
   3.4) Eingabe/Ausgabe-Einheit 36

4) Befehlsablauf im Mikrocomputer 38
   4.1) Struktur der Befehle 38
   4.2) Ablauf eines Befehlszyklus 40

5) Blockschaltbild eines Standard-System-Aufbaus eines Mikrocomputers mit dem Mikroprozessor 8080 46
   5.1) Zentraleinheit 46
   5.2) Speicher 57
   5.3) Eingabe/Ausgabe-Einheit 63

6) Programm-Abläufe 71
   6.1) "Normaler" sequentieller Programm-Ablauf 71
   6.2) Wartezustand 71
   6.3) Hold-Zustand 72
   6.4) Programm-Sprünge 73
   6.5) Unterprogramme 74
   6.6) Programm-Unterbrechungen 77
   6.7) Makros 82

7) Der Befehlssatz des Mikroprozessors 8080 85

8) Erstellen von Programmen für den Mikrocomputer 8080 107
   8.1) Assemblierung "von Hand" 107
   8.2) Assemblierung mittels Computer (Entwicklungssystem) 112

9) Der Mikroprozessor 8085 121
9.1) Einführung 121
9.2) Der Aufbau des Mikroprozessors 8085 125
9.3) Das Interrupt-System des Mikroprozessors 8085 129
9.4) Die seriellen Ein/Ausgabe-Leitungen SID und SOD 133
9.5) Der Befehlssatz des Mikroprozessors 8085 134

10) Wiederherstellung des vollständigen Adressbusses 136
10.1) Zeitlicher Ablauf beim Mikroprozessor 8080 und 8085 136
10.2) Erzeugung des vollständigen Adressbusses 139
10.3) Erzeugung der Steuersignale des 8080 aus dem 8085 140

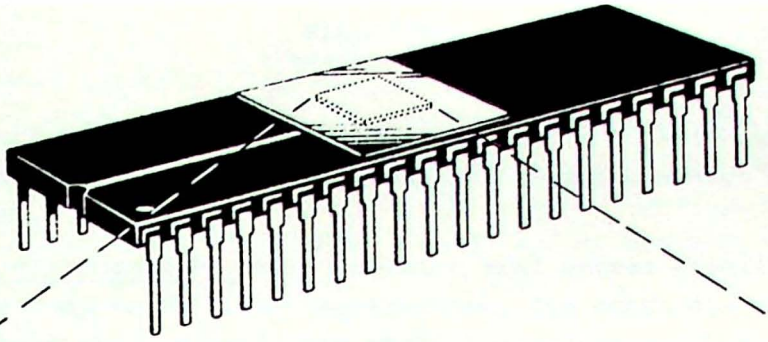
11) Aufbau von Mikrocomputern mit dem Mikroprozessor 8085 143
11.1) Mikrocomputersysteme mit Standard-Bausteinen 143
11.2) Ein Minimalsystem mit kombinierten Speicher-E/A-Bausteinen 8355 (8755) bzw. 8155 (8156) 148

12) Der Mikroprozessor Z80 158
12.1) Einführung 158
12.2) Der Aufbau des Mikroprozessors Z80 163
12.3) Zeitlicher Ablauf der Signale beim Mikroprozessor Z80 169
12.4) Das Interrupt-System des Mikroprozessors Z80 172

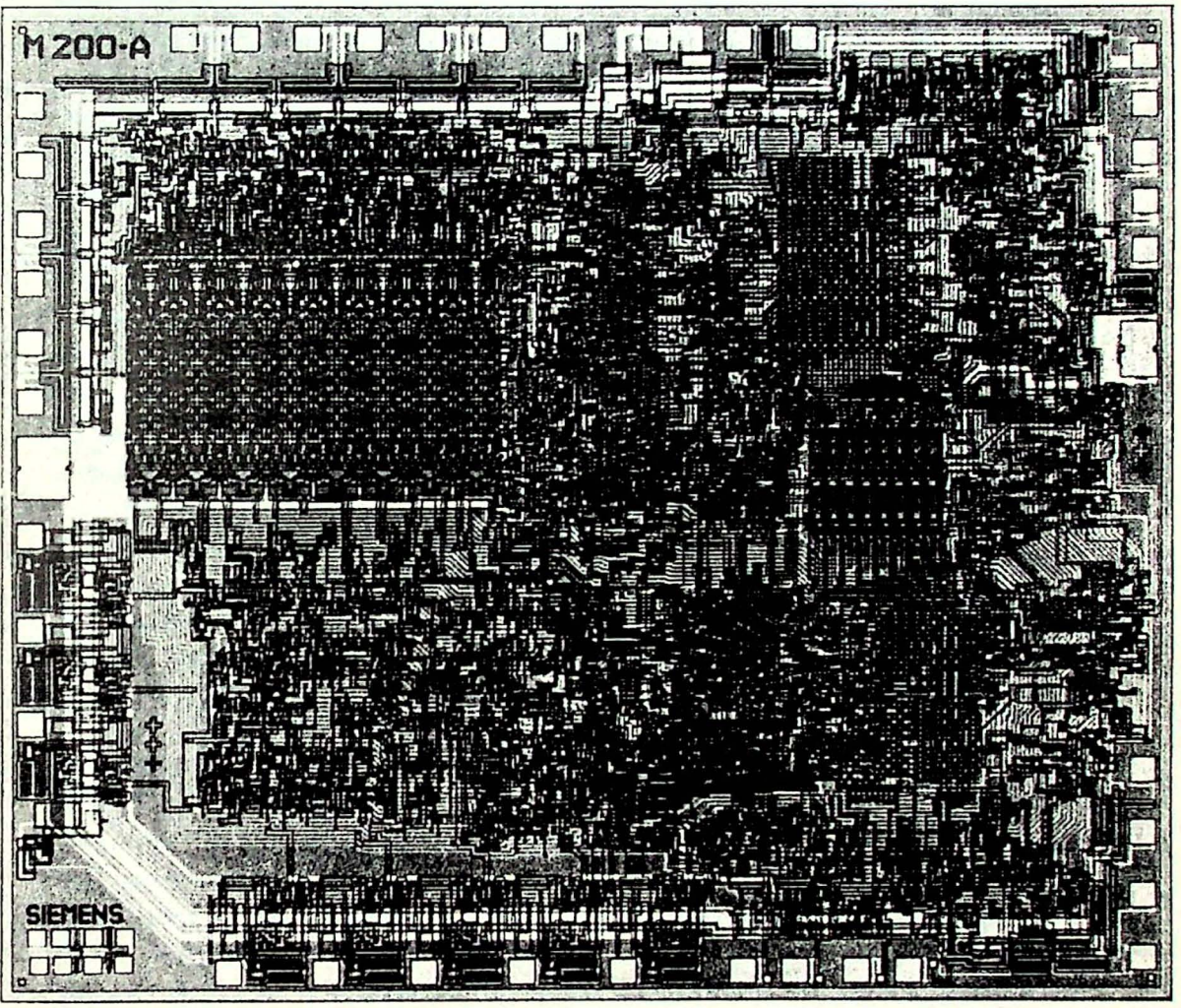
13) Der Befehlssatz des Mikroprozessors Z80 175
13.1) Einführung 175
13.2) Zusammenfassung der Z80-Befehle in Befehlsklassen 177
13.3) Alphabetische Zusammenstellung der Z80-Mnemonics 186
13.4) Die Flag-Beeinflussung durch die Z80-Befehle 192
13.5) Hexadezimale Reihenfolge der Z80-Befehle 193
13.6) Äquivalente Befehle 8080 - Z80 196

14) Aufbau von Mikrocomputern mit dem Mikroprozessor Z80 198
14.1) Eine Z80-CPU mit 8080-Steuerleitungen 198
14.2) Mikrocomputer mit Standard-E/A-Bausteinen 200
14.3) Spezielle E/A-Bausteine des Systems Z80 203
14.31) Der Parallel-E/A-Baustein Z80-PIO 203
14.32) Der Serien-E/A-Baustein Z80-SIO 207
14.33) Interrupt-Prioritätskette bei mehreren Bausteinen 210

Literaturverzeichnis 213



40-poliges Dual-In-Line-Gehäuse des Mikroprozessors 8080 (2x vergrößert).
Der Chip von 4x5 mm Größe befindet sich unter der mittleren Abdeckplatte.



Werksfoto SIEMENS

Halbleiterkristallplättchen (Chip) des Mikroprozessors SAB 8080 A
40x vergrößert. Die integrierte Schaltung enthält ca 5000 Transistoren.

EINLEITUNG

Dieses Buch handelt von Mikrocomputern, ihrer Funktionsweise und ihren universellen Einsatzmöglichkeiten in der gesamten digitalen Elektronik.

Was sind Mikrocomputer? Mikrocomputer sind extrem räumlich verkleinerte computerähnliche Logiksysteme, die durch den Fortschritt der modernen Mikrotechnologien möglich geworden sind. Der Stand der "VLSI-Technik" (Very Large-Scale Integration, Höchstintegration) erlaubt bei den integrierten Schaltkreisen den Aufbau von über hunderttausend Transistoren auf einem einzigen Silizium-Halbleiterkristallplättchen (Chip) von 5 x 5 mm^2, so daß die Funktionen eines leistungsfähigen Computers auf wenigen Chips realisiert werden können, im Extremfall sogar ein ganzer Computer auf einem einzigen Chip untergebracht werden kann. Solche, aus einem oder wenigen hochintegrierten Bausteinen aufgebaute Computer nennt man Mikrocomputer.

Was ist nun ein Mikroprozessor? Ein Mikroprozessor ist ein Teil eines Mikrocomputers und zwar ein sehr wichtiges Teil: Der Mikroprozessor ist im wesentlichen das Rechenwerk und das Steuerwerk des Mikrocomputers. Allein ist ein Mikroprozessor jedoch nicht arbeitsfähig, er muß durch Zusatzbausteine zu einem Mikrocomputer ausgebaut werden. Zusatzbausteine sind hauptsächlich:

1.) Speicher-Bausteine für das Programm und die laufenden Daten.
2.) Eingabe-Ausgabe-Bausteine, damit der Mikrocomputer mit der Außenwelt in Dialog treten kann.

Es besteht also ein begrifflicher Unterschied zwischen Mikrocomputer und Mikroprozessor. (Häufig meint man jedoch "Mikrocomputer", wenn man "Mikroprozessor" sagt).

Was kann man mit einem Mikroprozessor machen? Nur eines: Einen Mikrocomputer!

Was kann man mit einem Mikrocomputer machen? "Fast alles": Das gesamte Gebiet der digitalen Elektronik kann von Mikrocomputern abgedeckt werden. Der Einsatzbereich des Mikrocomputers ist also zum kleineren Teil das eigentliche "Rechnen". Der Mikrocomputer tritt an die Stelle der konventionellen "fest-verdrahteten"

digitalen Meß-Steuer-Regelschaltungen. Er ist dort universell verwendbar als steuerndes Logiksystem. Große Stückzahlen, sehr komplexe Problemstellungen und häufige Änderungen der Problemstellung machen den Einsatz von Mikrocomputern in der digitalen Elektronik zweckmäßig und rentabel. Darüber hinaus wird auch der Bereich der analogen Elektronik über Analog-Digital-Wandler und Digital-Analog-Rückwandler dem Einsatz von Mikrocomputern erschlossen.

Gegenüberstellung

Klassische digitale Elektronik ("fest-verdrahtete" digitale Elektronik)	Elektronik mit Mikrocomputern ("programmierte" digitale Elektronik)
Für jedes Problem der digitalen Elektronik (Messen, Steuern, Regeln) wird eine spezielle logische Schaltung entwickelt: Aus der Problemstellung wird (z.B. über Wahrheitstafeln, Maxterm-Minterm-Methode und Minimierung mit KV-Tafeln) die spezielle Schaltung mit logischen Grundbausteinen (UND, ODER, NICHT, NAND, NOR, Flipflops etc.) aufgebaut. Jede Änderung der Problemstellung hat eine neue Schaltung zur Folge!	Die bei der Lösung eines Problems zu vollziehenden logischen Schritte und Operationen werden von einem Mikrocomputer durchgeführt. Die Lösung des Problems, d.h. die Folge der auszuführenden logischen Schritte - das Programm - ist in einem Speicher niedergelegt; dieser Programmspeicher sagt dem Mikrocomputer, was er tun soll. Eine Änderung der Problemstellung hat "lediglich" eine Änderung des Anwenderprogramms zur Folge, das heißt, der Inhalt des Programmspeichers ("Software") wird ausgetauscht, während der Computer-Aufbau ("Hardware") weitgehend unverändert bleibt!

An die Stelle der individuell verdrahteten Schaltung tritt die individuelle Programmierung des universellen Mikrocomputers. Die Lösung des digitalen elektronischen Problems wird weitgehend auf das Programmieren verlagert.

Preisentwicklung

Das Erschließen immer neuer Anwendungsgebiete für Mikrocomputer läßt die Produktionsziffern von Mikroprozessoren und Mikrocomputern steil in die Höhe gehen. Hohe Stückzahlen bewirken fallende Preise, niedrigere Preise mobilisieren neue Anwender und eröffnen neue Anwendungsgebiete, und dies bedingt weiter steigende Stückzahlen und weiter fallende Preise.
Während 1976 etwa 3 Millionen Mikroprozessoren hergestellt wurden, belaufen sich die Produktionsziffern 1983 weltweit bereits auf annähernd 1 Milliarde Stück. Ein 8-Bit-Mikroprozessor kostet heute nur noch ca. DM 10.--, ein 8-Bit-Mikrocomputer in Minimalkonfiguration etwa hundert D-Mark.

Kurze Geschichte der digitalen Rechenmaschinen

Erste Rechenmaschine (4 Spezies): Wilhelm Schickard 1623 (s.S.157)

(Geboren 1592 in Herrenberg. Professor für Hebräisch und Aramäisch an der Universität Tübingen, später für Astronomie, Mathematik und Geodäsie. Gestorben 1635 an der Pest in Tübingen.)

Erster Entwurf einer programmgesteuerten (mechanischen) Rechenmaschine: Ch. Babbage 1833.

Bis 1940 wurden wahre Wunderwerke von mechanischen Rechenmaschinen konstruiert.

Erste elektrische, programmgesteuerte Maschine: K.Zuse 1941:

Z3, Relais-Rechner mit 2000 Relais.

(etwa gleichzeitig: H.H.Aiken, Relais-Rechner ohne Programmsteuerung: MARK I)

Erste elektronische Maschine: J.P.Eckert, J.W.Mauchly 1945/46:

ENIAC mit 20 000 Röhren, nicht programmgesteuert.

Erster kommerziell vertriebener Computer: J.P.Eckert, J.W.Mauchly

1950: UNIVAC I, mit Röhren, speicherprogrammiert.

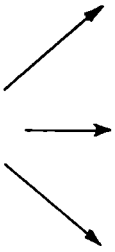
Erster Transistor-Computer: J.H.Felker 1955: TRADIC.

(1958 erster volltransistorisierter Computer für allgemeinen Markt: SIEMENS 2002 mit Ge-Transistoren).

Die weitere Entwicklung der Computer verlief in drei Richtungen -- Groß-Computer, Mini-Computer, Mikro-Computer --, wobei die Abgrenzung dieser drei Computer-Arten gegeneinander mehr oder weniger willkürlich ist:

"Groß-Computer": Zur Lösung sehr komplexer Rechenprobleme und zur Verarbeitung großer Datenmengen (EDV). Wortlänge 32 - 64 Bit.

Verzweigung: (unscharf)



"Mini-Computer": Kompakte Computer in diskreter Technik für Prozeßrechneranwendungen und mittlere Datenmengen. Wortlänge 12 - 32 Bit.
1965 erstes Gerät von der Firma DIGITAL EQUIPMENT: PDP 8.

"Mikro-Computer": Aus hochintegrierten Logikbausteinen aufgebaute Computer kleinster Bauweise. Wortlänge 4 - 16 Bit (32 Bit).
1971 erste Mikroprozessoren von der Firma INTEL: 4004, 8008.

Anwendungsgebiete für Mikrocomputer

Aus den ständig wachsenden Anwendungsgebieten seien einige beispielhafte Bereiche herausgegriffen:

Nachrichtentechnik:
Fernschreiber, Fernkopierer, elektronisches Fernsprechvermittlungssystem, Bildfernsprechen, Infrarot-Nachrichtenübertragung, Kommunikation in Breitbandkabelnetzen, Videotext, Bildschirmtext, Verkehrsfunk-Decoder, Rundsichtradar, Funksprechgeräte, Spracherkennung.

"intelligente" Meßgeräte:
Oszilloskope, Spektrum-Analysatoren, Bauelemente-Tester, Sortierlogiken, rechnende Waagen, Digitalmultimeter, Meßbrücken, "smarte" Meßinstrumente (Meßinstrumente, bei denen keine Fehlbedienung möglich ist), Dosimetriesysteme, Signalsynthesizer, Logik-Analysatoren.

Physikalische und chemische Geräte:
Überwachung von Reaktoren, Beschleunigern, Pattern-Analyzer, Massenspektrometer, Gaschromatographen, Analyseautomaten, Aufdampfanlagen, Warnanlagen für Umweltschutz, Klimaanlagen.

Medizin, Biologie:
Diagnosegeräte, Laborauswertungen, Patientenüberwachung, Röntgengerätesteuerung, psychologische Testsysteme, Verwaltungscomputer für Praxen.

Industrielle Steuerung:
Prozeßsteuerung und -kontrolle, Aufzugsteuerung, dezentrale Regelsysteme (Subsysteme), Roboter (Manipulatoren), NC-Maschinen-Steuerung, Produktionskontrolle, Pressensteuerung, Fließbandregelung, Zentraleinheit für Regelkreise, Testsystem für IC und Leiterplatten, Verkehrssteuerung und -lenkung.

Computer-Peripherie und Datenverarbeitung:
Datensichtgeräte, Kartenleser, Floppy-Disks, Bandgeräte (Kassetten), Drucker, Klarschriftleser, "intelligente" Terminals (Eingabegeräte mit Plausibilitätskontrolle), Datenkonzentratoren (Daten- "Vor"-Verarbeitung, Datensammlung), Geräte der mittleren Datentechnik, Personal-Computer, Speicher-, I/O-Management, Multiprocessing (System mit mehreren Mikroprozessoren), Emulation von Kleinrechnern, Taschenrechner, Bankanlagen (automatisches Quittieren von Schecks u.ä.), allgemeine Bürotechnik (Organisation einer Kartei, automatische Schreibmaschinen, Kopiermaschinen).

Dezentrale Bordcomputer:
In Flugzeugen, Schiffen, Kraftfahrzeugen, Waffensystemen, Raketensteuerungen (Raumfahrt).

Kfz-Elektronik:
Motorsteuerung zur Optimierung des Kraftstoffverbrauchs, Antiblockier-Bremssystem, Antikollisionsradar, Bord-Computer, Überwachung aller wichtigen Funktionen, Kfz-Diagnosegeräte.

Unterhaltungs- und Freizeitelektronik:
Rundfunk- und Fernsehgeräteelektronik, Plattenspieler- und Bandgerätesteuerung, Videorecorder, Fernsteuerung, Fernsehspiele, Filmkamera und -projektor, elektronische Brettspiele, Tonsynthese in Orgeln, Heimcomputer, Spielautomaten, Spielzeuge.

Haushaltselektronik:
Waschmaschinen, Wäschetrockner, Geschirrspüler, Nähmaschinen-Programmierung, Heizungs- und Lichtregelung, Raumüberwachung.

Zum methodischen Aufbau dieses Buches:

Die Verschiedenheit der heute verfügbaren Typen von Mikroprozessoren, die sich nicht nur in ihrer Architektur, sondern auch in ihrer Sprache (d.h. in ihrem Befehlssatz) und in ihren Programmiermöglichkeiten unterscheiden, macht eine allgemeine systematische Darstellung der Mikrocomputer und Mikroprozessoren sehr schwierig. Es wird daher hier der Weg beschritten, den Aufbau und die Funktionsweise von Mikrocomputern an Hand eines exemplarischen Beispiels zu erläutern. Als exemplarischer Vertreter wurde der Mikroprozessor 8080 ausgewählt, der klassische 8-Bit-Mikroprozessor, der einen Markstein in der Entwicklung der Mikrocomputer darstellt.
Es werden also im ersten Teil dieses Buches Mikrocomputer des Systems 8080 in ihrem Aufbau, ihrer Funktionsweise, ihrem Befehlssatz und in ihren Programmiermöglichkeiten eingehend dargestellt mit der Hoffnung, daß danach auch das Einarbeiten in andere Mikrocomputer-Typen keine grundsätzlichen Schwierigkeiten mehr macht. Kurz gesagt: Es soll die "Fahrschule" für Mikrocomputer auf dem Typ 8080 gemacht werden und damit die Voraussetzung für das Umsteigen auf einen anderen Typ gegeben sein.
Im zweiten Teil des Buches werden die aus dem Typ 8080 weiterentwickelten Mikroprozessoren 8085 und Z80 vorgestellt. Der detaillierte Aufbau von Mikrocomputern mit diesen Mikroprozessoren wird an Hand der beim 8080 erworbenen Kenntnisse erläutert, und der Leser wird so mit zwei der modernsten und leistungsfähigsten 8-Bit-Mikrocomputersysteme vertraut gemacht, die heute auf dem Markt angeboten werden.

1) DARSTELLUNG VON INFORMATION DURCH BITMUSTER

1.1) Die Begriffe Bit, Byte, Baud

Mikrocomputer sind digital-elektronische Schaltungen, welche Information nur durch 2-wertige Zustandsgrößen ("binär") darstellen. Die beiden Zustände werden mit 1 und O bezeichnet, bzw. bei Realisierung durch Spannungen mit H (High) und L (Low). Einen solchen 2-wertigen Zustand nennt man

1 Bit (= 1 Binary Digit = 1 Binärziffer).

Gleichzeitig ist 1 Bit die Einheit für binäre Information. Das heißt also zum Beispiel, daß eine "8-Bit-Information" aus 8 Stellen mit jeweils O/1-Alternative besteht, etwa: 10010110 .
Ist an einer Stelle die 1 vorhanden, sagt man: "Das Bit ist gesetzt". Im anderen Fall, bei O: "Das Bit ist rückgesetzt".

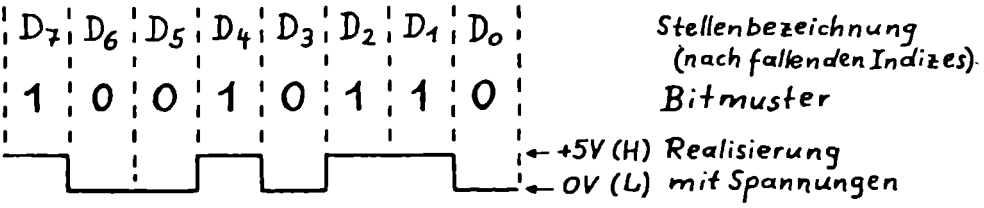
Größere Einheiten für Information sind:

8 Bit = 1 Byte (gelegentlich: 4 Bit = 1 Nibble
16 Bit = 1 Wort)
2^{10} = 1024 = 1 K (sprich: "ka" nicht "kilo"!)
2^{20} = 1 048 576 = 1 M

also z.B.: 8 K Bit = 8x1024 Bit = 8192 Bit = 1024 Byte = 1 K Byte

Eine längere Information wird aus mehreren kurzen "Datenworten" zusammengesetzt, und man nennt die Zahl der Bits dieser Datenworte (z.B. 4, 8, 16 Bit) die "Wortlänge" des Datenwortes.

Beispiel: Datenwort mit 8 Bit Länge (1 Byte): 10010110



Die Darstellung von Information durch 2-wertige Bitmuster hat den Vorteil, daß diese Bitmuster elektronisch leicht realisiert werden können, daß sie sicher zu unterscheiden sind und störungsunempfindlich übertragen werden können.

Wird binäre Information seriell (nacheinander) übertragen, so ist für die Übertragungsgeschwindigkeit die Zahl der übertragenen Bits pro Sekunde maßgebend: "Datenübertragungsrate" oder "Baud-Rate" (nach dem französischen Nachrichtentechniker Baudot (1845-1903)).

Die Einheit der Datenübertragungsrate ist

1 Baud = 1 Bd = 1 Bit/s

Übliche Baudraten sind: 150-300-600-1200-2400-4800-9600-19200 Bd, eine bei Fernschreibern oft verwendete Übertragungsrate ist 110 Bd.

1.2) Darstellung von Zahlen durch Bitmuster

1.21) Zahlensysteme

Für die Erklärung des Aufbaus von Zahlensystemen wird das Dezimalsystem zum Vergleich herangezogen:

Das Dezimalsystem (D) besitzt 10 Elementarzeichen 0,1,2,...9 und eine Anordnung der Ziffern nach fallenden Potenzen von 10:

$$1979 = 1 \cdot 10^3 + 9 \cdot 10^2 + 7 \cdot 10^1 + 9 \cdot (10^0)$$

a) Duales Zahlensystem oder Dual-System (B)

2 Zeichen: 0,1

Anordnung der Zeichen nach fallenden Potenzen von 2

Zusammenhang zwischen Dezimal- und Dualzahlen:

Dezimal (D)	0	1	2	3	4	5	6	7
Dual (B)	0	1	10	11	100	101	110	111

8	9	10	11	12	13	14	15	16
1000	1001	1010	1011	1100	1101	1110	1111	10000

(Lies z.B. bei 100(B): "Eins-Null-Null", nicht "Hundert"!)

Beispiel:

Dual: 1 0 0 1 0 1 1 0 (B)

$(2^7 \; 2^6 \; 2^5 \; 2^4 \; 2^3 \; 2^2 \; 2^1 \; 2^0)$

Dezimal: $1 \cdot 2^7 + 1 \cdot 2^4 + 1 \cdot 2^2 + 1 \cdot 2 = 150$ (D)

Die Dualzahl-Darstellung ergibt also ein reines Bitmuster aus Nullen und Einsen und ist daher sehr geeignet für Computer. Der Nachteil des Dual-Systems ist seine relative Unübersichtlichkeit für den Menschen bei größeren Zahlen.
Daher werden zur besseren Übersichtlichkeit Bitgruppen zusammengefaßt:

Zusammenfassen von je 3 Bit ergibt das Oktalsystem
Zusammenfassen von je 4 Bit ergibt das Hexadezimalsystem

b) Oktal-System (Q)

8 Zeichen: O,1,2,3,4,5,6,7
Anordnung der Zeichen nach fallenden Potenzen von 8

Zusammenhang zwischen Dezimal- und Oktalzahlen:

Dezimal (D)	O	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Oktal (Q)	O	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20

(Lies z.B. bei 12(Q): "Eins-zwei", nicht "zwölf"!)

Beispiel:

Dual: (O)11 O1O 111 (B)
Oktal: 3 2 7 (Q)
Dezimal: $3 \cdot 8^2 + 2 \cdot 8^1 + 7 \cdot 8^O = 215(D)$

c) Hexadezimal-System oder Sedezimal-System (H)

Übliches System bei der Programmierung von Mikrocomputern.

16 Zeichen: O,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
Anordnung der Zeichen nach fallenden Potenzen von 16

Zusammenhang zwischen Dezimal- und Hexadezimalzahlen:

Dezimal (D)	O	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Hexa-dezimal (H)	O	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13

Beispiel:

Dual: 1101 0111 (B)
Hexadezimal: D 7 (H)
Dezimal: $13.16^1 + 7 \cdot 16^0 = 215$(D)

Für das Programmieren eines Mikrocomputers sagt man dann bequemerweise etwa F1B9(H), und die Maschine interpretiert dies als Bitmuster 1111 0001 1011 1001(B).

d) Umwandlung von Zahlensystemen

Umwandlung von Dezimalzahlen in Dualzahlen (das hier verwendete Prinzip gilt entsprechend für jede andere Umwandlung):

Beispiel: 30(D) = 11110(B)
Die Umwandlung geschieht nach folgendem Prinzip:

```
30 : 2 = 15 Rest 0 ↑
15 : 2 =  7 Rest 1 |   Die Reste von unten nach oben
 7 : 2 =  3 Rest 1 |   gelesen ergeben die gesuchte
 3 : 2 =  1 Rest 1 |   Dualzahl
 1 : 2 =  0 Rest 1 |
```

Umwandlung von Dezimalbrüchen in Dualbrüche (das Prinzip gilt allgemein):

Beispiel: 0,325(D) = 0,010 $\overline{1001}$...(B)
Die Umwandlung geschieht nach folgendem Prinzip:

```
0,325 · 2 = 0,65   Ziffer vor dem Komma 0 |
0,65  · 2 = 1,3    Ziffer vor dem Komma 1 |
0,3   · 2 = 0,6            -          0  |  Die Ziffern vor dem
0,6   · 2 = 1,2            -          1  |  Komma von oben nach
0,2   · 2 = 0,4            -          0  |  unten gelesen erge-
0,4   · 2 = 0,8            -          0  |  ben die gesuchte
0,8   · 2 = 1,6            -          1  |  Dualdarstellung
0,6   · 2 = 1,2            -          1  ↓
```

Rückwandlung vom Dualsystem ins Dezimalsystem (das Prinzip gilt auch für andere Zahlensysteme):

Beispiel: $11110{,}010\overline{1001}....(B) = 30{,}325(D)$

Die Umwandlung geschieht nach folgendem Prinzip:

(B):	1	1	1	1	0 ,	0	1	0	1	0	0	1	
Wert :	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	
(D):	16 +	8 +	4 +	2		+ 0,25	+	0,0625	+	0,00781	25		

= 30,3203125 (Die Abweichung entsteht wegen des abgebrochenen unendlichen Bruchs)

1.22) Darstellung von negativen Zahlen durch Bitmuster

Es sei ein Datenwort von 8 Bit Länge gegeben (übliche Wortlänge bei Mikrocomputern). Damit sind nach der Kombinatorik $2^8 = 256$ verschiedene Zahlen darstellbar. Will man positive und negative Zahlen darstellen, so bräuchte man z.B. lediglich das vorderste Bit als Vorzeichenbit zu definieren: 0 als vorderstes Bit bedeutet positive Zahl, 1 als vorderstes Bit bedeutet negative Zahl. Die verbleibenden 7 Bit stellen dann als Dualzahl den Betrag der Zahl dar (zwischen 0 und 127).

Für den Mikrocomputer geeigneter ist jedoch folgende Darstellung:

Positive Zahlen: von +0 bis +127: Vorderstes Bit wird 0 gesetzt, die restlichen 7 Bit stellen als Dualzahl den Wert der Zahl dar (0 bis 127).

Negative Zahlen: von -1 bis -128: Es wird anstelle der negativen Zahl A das "Zweierkomplement" der positiven Zahl A genommen: $(2^8 - A)$. Das Zweierkomplement der Zahl A wird schematisch wie folgt gebildet: Jedes Bit der Dualzahl A wird invertiert und dann zum Ergebnis noch 1 hinzuaddiert.

Beispiel: Darstellung der Zahl -53(D):

Es ist +53 als Dualzahl:	0011 0101
Jedes Bit invertiert ergibt:	1100 1010
1 hinzuaddiert :	+1
	1100 1011

Dies ist die Darstellung der Zahl -53(D) als Zweierkomplement.

Vorteile dieser Darstellungsart: 1.) alle negativen Zahlen beginnen mit 1 im vordersten Bit, 2.) die Subtraktion wird in eine Addition übergeführt: $B - A = B + \boxed{2^8 - A}$ (-2^8).

Beispiel:	+6 :	0000 0110	
Berechne 15-6	invertiert :	1111 1001	
		+1	
	ergibt -6 :	1111 1010	Zweierkomplement
	dazu addiert +15 :	+0000 1111	
	Ergebnis :	(1)0000 1001	(-2^8)

Der Überlauf wird unterschlagen, es bleibt 0000 1001(B) = 9(D). Damit erhält man das richtige Ergebnis. (Der Überlauf gibt Auskunft über das Vorzeichen des Ergebnisses).

1.23) Reelle Zahlen in Gleitkomma-Darstellung als Bitmuster

Will man große Wertebereiche von reellen Zahlen mit Computern erfassen, so bedient man sich der Darstellung der Dualzahlen mit "Gleitkomma". Die Gleitkommadarstellung ist in den Mikrocomputern nicht "hardewaremäßig" eingebaut, sie muß bei Bedarf durch entsprechende Programmierung ("Unterprogramme") erfolgen. Es wird deshalb nur kurz auf diese Darstellungsform eingegangen und der Anwender auf die Literatur verwiesen.

Eine Dualzahl in Gleitkommadarstellung hat die Form

$$\text{Mantisse} \times 2^{\text{Exponent}}$$

wobei dafür gesorgt wird, daß die Mantisse immer im Zahlenbereich zwischen 0,5(D) und < 1(D) liegt ("Normalisierte Mantisse"). Die Mantisse läßt sich ja immer in diesen Zahlenbereich bringen, indem der Exponent entsprechend verändert wird. Es werden 3 Bytes zur Dualdarstellung der Mantisse (24 Dualstellen) und 1 Byte zur Darstellung des Dualexponenten (7 Dualstellen) und des Vorzeichens (1 Stelle) verwendet.

Beispiel: Die Dezimalzahl +19,0096 soll als Dualzahl in Gleitkommadarstellung geschrieben werden.
Durch Division mit $2^5 = 32$ läßt sich diese Zahl zunächst in den gewünschten Mantissenbereich zwischen 0,5 und 1 überführen:
$19{,}0096 = 0{,}59405 \times 2^5$.

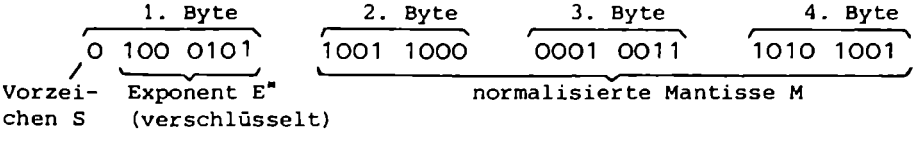
Jetzt muß die Mantisse 0,59405 in eine Dualzahl umgewandelt werden:

M = 0,59405(D) = 0,1001 1000 0001 0011 1010 1001 ...(B)

Das Vorzeichen der Mantisse wird im vordersten Bit des Exponenten-Byte niedergelegt: 0 bedeutet positive Zahl,
1 bedeutet negative Zahl.

Die Darstellung des Exponenten erfolgt verschlüsselt, damit auch negative Exponenten mit erfaßt werden können: Anstelle des Exponenten E wird im Exponenten-Byte der Wert 64 + E dargestellt. Damit lassen sich in 7 Dualstellen (0 - 127) die Exponenten von -64 bis +63 unterbringen. Der tatsächliche Exponent unseres Beispiels, nämlich 5, wird also verschlüsselt dargestellt als $E^* = 64 + 5 = 69$, in dualer Schreibweise $E^* = 100\ 0101(B)$.

Die Mantisse und der verschlüsselte Exponent werden nun wie folgt in den 4 Bytes der Gleitkommadarstellung abgelegt:



Dies ist also die Gleitkommadarstellung der Zahl + 19,0096(D).

Der Wert der so dargestellten Zahl lautet explizit:

$$(-1)^S \cdot 2^{(E^*-64)} \cdot M$$

Diese Gleitkommadarstellung kann in ähnlicher Form im Mikrocomputer 8080 programmiert werden.

1.3) Darstellung von Zeichen durch Bitmuster

Da der Mikrocomputer nur 0 und 1 "versteht", müssen auch Zeichen und Texte, die man dem Mikrocomputer mitteilen will, als Bitmuster kodiert werden.
Alle Mikrocomputer verwenden als Zeichencode den ASCII-Code ("American Standard Code for Information Interchange"), einen ursprünglich 8-Bit-Fernschreibcode aus den USA.
Die Mikrocomputer kommen mit 7 Bit aus, das 8. Bit (vorderste Bit) wird entweder Null gesetzt oder als Paritätsbit verwendet.
Im Mikrocomputersystem 8080 wird ein 7-Bit-ASCII-Code verwendet, der dem Standard 8-Bit-ASCII-Code mit Null gesetztem höchstwertigem Bit entspricht. Die nachfolgende Tabelle enthält die Bitmuster für die einzelnen ASCII-Zeichen und die für den Operateur bequemere (weil übersichtlichere) Darstellung im Hexadezimalcode.

Beispiel:

Text:	E	K	A
zugehöriges Bitmuster:	1000101	1001011	1000001
hexadezimaler Code :	45	4B	41

Steuerzeichen		Binär-Code	Hexadezimaler Code
NUL	– Null	000 0000	00
SOH	– Anfang des Kopfes	000 0001	01
STX	– Anfang des Textes	000 0010	02
ETX	– Ende des Textes	000 0011	03
EOT	– Ende der Übertragung	000 0100	04
ENQ	– Stationsaufforderung	000 0101	05
ACK	– Positive Rückmeldung	000 0110	06
BEL	– Klingelzeichen	000 0111	07
BS	– Rückwartsschritt	000 1000	08
HT	– Horizontal-Tabulator	000 1001	09
LF	– Zeilenvorschub	000 1010	0A
VT	– Vertikal-Tabulator	000 1011	0B
FF	– Formularvorschub	000 1100	0C
CR	– Wagenrücklauf	000 1101	0D
SO	– Dauerumschaltung	000 1110	0E
SI	– Rückschaltung	000 1111	0F
DLE	– Datenübertragungsumschaltung	001 0000	10
DC1	– Gerätesteuerung 1	001 0001	11
DC2	– Gerätesteuerung 2	001 0010	12
CC3	– Gerätesteuerung 3	001 0011	13
DC4	– Gerätesteuerung 4	001 0100	14
NAK	– Negative Rückmeldung	001 0101	15
SYN	– Synchronisierung	001 0110	16
ETB	– Blockübertragungsende	001 0111	17
CAN	– Ungültig	001 1000	18
EM	– Ende der Aufzeichnung	001 1001	19
SUB	– Substitution	001 1010	1A
ESC	– Umschaltung	001 1011	1B
FS	– Datentrennung	001 1100	1C
GS	– Gruppentrennung	001 1101	1D
RS	– Blocktrennung	001 1110	1E
US	– Teilgruppentrennung	001 1111	1F
DEL	– Löschen	111 1111	7F

TABELLE: ASCII STEUERZEICHENCODE FÜR FERNSCHREIBER

ASCII-Code (American Standard Code for Information Interchange)

Zeichen	Binär Code	Hexadezimaler Code	Zeichen	Binär-Code	Hexadezimaler Code
Leerspalte blank	010 0000	20	P	101 0000	50
!	010 0001	21	Q	101 0001	51
" (doppelter Anführungsstrich)	010 0010	22	R	101 0010	52
#	010 0011	23	S	101 0011	53
$	010 0100	24	T	101 0100	54
%	010 0101	25	U	101 0101	55
&	010 0110	26	V	101 0110	56
' (Anführungsstrich)	010 0111	27	W	101 0111	57
(	010 1000	28	X	101 1000	58
)	010 1001	29	Y	101 1001	59
* (Stern)	010 1010	2A	Z	101 1010	5A
+	010 1011	2B	[	101 1011	5B
, (Komma)	010 1100	2C	\	101 1100	5C
– (Minus)	010 1101	2D	]	101 1101	5D
. (Punkt)	010 1110	2E	Λ (↑)	101 1110	5E
/	010 1111	2F	– (Unterstreichung) (←)	101 1111	5F
0	011 0000	30	`	110 0000	60
1	011 0001	31	a	110 0001	61
2	011 0010	32	b	110 0010	62
3	011 0011	33	c	110 0011	63
4	011 0100	34	d	110 0100	64
5	011 0101	35	e	110 0101	65
6	011 0110	36	f	110 0110	66
7	011 0111	37	g	110 0111	67
8	011 1000	38	h	110 1000	68
9	011 1001	39	i	110 1001	69
:	011 1010	3A	j	110 1010	6A
;	011 1011	3B	k	110 1011	6B
<	011 1100	3C	l	110 1100	6C
=	011 1101	3D	m	110 1101	6D
>	011 1110	3E	n	110 1110	6E
?	011 1111	3F	o	110 1111	6F
@	100 0000	40	p	111 0000	70
A	100 0001	41	q	111 0001	71
B	100 0010	42	r	111 0010	72
C	100 0011	43	s	111 0011	73
D	100 0100	44	t	111 0100	74
E	100 0101	45	u	111 0101	75
F	100 0110	46	v	111 0110	76
G	100 0111	47	w	111 0111	77
H	100 1000	48	x	111 1000	78
I	100 1001	49	y	111 1001	79
J	100 1010	4A	z	111 1010	7A
K	100 1011	4B	{	111 1011	7B
L	100 1100	4C	|	111 1100	7C
M	100 1101	4D	}	111 1101	7D
N	100 1110	4E	~	111 1110	7E
O	100 1111	4F			

TABELLE: ASCII ZEICHENCODE

1.4) Darstellung von Maschinen-Befehlen durch Bitmuster

Die Befehle, mit denen dem Mikrocomputer mitgeteilt wird, was er zu tun hat, werden vom Anwender in Form eines Programmes zusammengestellt und dann in einem Programmspeicher des Mikrocomputers "nichtflüchtig" niedergelegt. Da auch der Programmspeicher wie der übrige Mikrocomputer binär arbeitet, müssen die Befehle als binär kodiertes Bitmuster vorliegen. Für jeden Mikroprozessor-Typ gibt es einen charakteristischen Befehlssatz, in welchem jedem einzelnen Befehl ein bestimmtes Bitmuster zugeordnet ist: Die sogenannte Maschinensprache des Mikroprozessors.

Beim Mikroprozessor Typ 8080 bestehen die einzelnen Befehle aus "Worten" von 8 Bit Länge. Mit 8-Bit-Worten sind $2^8 = 256$ verschiedene Befehle darstellbar als Bitmuster. Der Befehlssatz des Mikroprozessors 8080 nützt hiervon 244 aus (siehe Tabelle).

Die Darstellung der Befehle als 8-stellige Bitmuster ist für den Operateur unhandlich; er faßt die 8-Bit-Worte bequemerweise wieder in Hexadezimalzahlen zusammen. Zum leichteren Merken der einzelnen Befehle werden darüber hinaus einprägsame Abkürzungen der Befehle eingeführt ("mnemonische" Abkürzungen), die sogenannte ASSEMBLER-Sprache des Mikroprozessors.

Beispiel: Ein vorkommender Befehl an den Mikrocomputer lautet:
"Bringe den Inhalt des Registers A des Mikroprozessors ins Register B!"

Das Bitmuster dieses Befehls lautet in der Maschinensprache des Mikroprozessors 8080:	0100 0111
Die zugehörige hexadezimale Codierung lautet:	4 7
Die mnemonische Abkürzung in der ASSEMBLER-Sprache lautet:	MOV B,A

In der nachfolgenden Tabelle sind die 244 Befehle des Mikroprozessors 8080 in hexadezimaler Schreibweise und in mnemonischer Abkürzung zusammengestellt. Die Tabelle ist "als erstes Kennenlernen" gedacht, die einzelnen Befehle werden später ausführlich abgehandelt werden.

	MNEMONIC		MNEMONIC		MNEMONIC		MNEMONIC		MNEMONIC		MNEMONIC
00	NOP	2B	DCX H	56	MOV D,M	81	ADD C	AC	XRA H	D7	RST 2
01	LXI B,D16	2C	INR L	57	MOV D,A	82	ADD D	AD	XRA L	D8	RC
02	STAX B	2D	DCR L	58	MOV E,B	83	ADD E	AE	XRA M	D9	--
03	INX B	2E	MVI L,D8	59	MOV E,C	84	ADD H	AF	XRA A	DA	JC Adr
04	INR B	2F	CMA	5A	MOV E,D	85	ADD L	B0	ORA B	DB	IN D8
05	DCR B	30	-- (SIM)	5B	MOV E,E	86	ADD M	B1	ORA C	DC	CC Adr
06	MVI B,D8	31	LXI SP,D16	5C	MOV E,H	87	ADD A	B2	ORA D	DD	--
07	RLC	32	STA Adr	5D	MOV E,L	88	ADC B	B3	ORA E	DE	SBI D8
08	--	33	INX SP	5E	MOV E,M	89	ADC C	B4	ORA H	DF	RST 3
09	DAD B	34	INR M	5F	MOV E,A	8A	ADC D	B5	ORA L	E0	RPO
0A	LDAX B	35	DCR M	60	MOV H,B	8B	ADC E	B6	ORA M	E1	POP H
0B	DCX B	36	MVI M,D8	61	MOV H,C	8C	ADC H	B7	ORA A	E2	JPO Adr
0C	INR C	37	STC	62	MOV H,D	8D	ADC L	B8	CMP B	E3	XTHL
0D	DCR C	38	--	63	MOV H,E	8E	ADC M	B9	CMP C	E4	CPO Adr
0E	MVI C,D8	39	DAD SP	64	MOV H,H	8F	ADC A	BA	CMP D	E5	PUSH H
0F	RRC	3A	LDA Adr	65	MOV H,L	90	SUB B	BB	CMP E	E6	ANI D8
10	--	3B	DCX SP	66	MOV H,M	91	SUB C	BC	CMP H	E7	RST 4
11	LXI D,D16	3C	INR A	67	MOV H,A	92	SUB D	BD	CMP L	E8	RPE
12	STAX D	3D	DCR A	68	MOV L,B	93	SUB E	BE	CMP M	E9	PCHL
13	INX D	3E	MVI A,D8	69	MOV L,C	94	SUB H	BF	CMP A	EA	JPE Adr
14	INR D	3F	CMC	6A	MOV L,D	95	SUB L	C0	RNZ	EB	XCHG
15	DCR D	40	MOV B,B	6B	MOV L,E	96	SUB M	C1	POP B	EC	CPE Adr
16	MVI D,D8	41	MOV B,C	6C	MOV L,H	97	SUB A	C2	JNZ Adr	ED	--
17	RAL	42	MOV B,D	6D	MOV L,L	98	SBB B	C3	JMP Adr	EE	XRI D8
18	--	43	MOV B,E	6E	MOV L,M	99	SBB C	C4	CNZ Adr	EF	RST 5
19	DAD D	44	MOV B,H	6F	MOV L,A	9A	SBB D	C5	PUSH B	F0	RP
1A	LDAX D	45	MOV B,L	70	MOV M,B	9B	SBB E	C6	ADI D8	F1	POP PSW
1B	DCX D	46	MOV B,M	71	MOV M,C	9C	SBB H	C7	RST 0	F2	JP Adr
1C	INR E	47	MOV B,A	72	MOV M,D	9D	SBB L	C8	RZ	F3	DI
1D	DRC E	48	MOV C,B	73	MOV M,E	9E	SBB M	C9	RET	F4	CP Adr
1E	MVI E,D8	49	MOV C,C	74	MOV M,H	9F	SBB A	CA	JZ Adr	F5	PUSH PSW
1F	RAR	4A	MOV C,D	75	MOV M,L	A0	ANA B	CB	--	F6	ORI D8
20	-- (RIM)	4B	MOV C,E	76	HLT	A1	ANA C	CC	CZ Adr	F7	RST 6
21	LXI H,D16	4C	MOV C,H	77	MOV M,A	A2	ANA D	CD	CALL Adr	F8	RM
22	SHLD Adr	4D	MOV C,L	78	MOV A,B	A3	ANA E	CE	ACI D8	F9	SPHL
23	INX H	4E	MOV C,M	79	MOV A,C	A4	ANA H	CF	RST 1	FA	JM Adr
24	INR H	4F	MOV C,A	7A	MOV A,D	A5	ANA L	D0	RNC	FB	EI
25	DCR H	50	MOV D,B	7B	MOV A,E	A6	ANA M	D1	POP D	FC	CM Adr
26	MVI H,D8	51	MOV D,C	7C	MOV A,H	A7	ANA A	D2	JNC Adr	FD	--
27	DAA	52	MOV D,D	7D	MOV A,L	A8	XRA B	D3	OUT D8	FE	CPI D8
28	--	53	MOV D,E	7E	MOV A,M	A9	XRA C	D4	CNC Adr	FF	RST 7
29	DAD H	54	MOV D,H	7F	MOV A,A	AA	XRA D	D5	PUSH D		
2A	LHLD Adr	55	MOV D,L	80	ADD B	AB	XRA E	D6	SUI D8		

Befehlsliste des Mikroprozessors 8080/85

in der Reihenfolge der hexadezimalen Abkürzungen.
(Die Befehle RIM (20) und SIM (30) sind nur beim Mikroprozessor 8085 anwendbar)

1.5) Beispiel für die Interpretation eines 8 Bit-Datenwortes

Da ein Mikrocomputer nur 0 und 1 versteht, kann ein Bitmuster von 8 Bit Länge, das dem Computer mitgeteilt wird, zunächst für die Maschine verschiedenes bedeuten: Eine Zahl, ein ASCII-Zeichen oder ein Maschinenbefehl.

Der Mikrocomputer muß also instruiert werden, ob er das ihm vorgelegte Bitmuster nun als Zahl, als Zeichen oder als Befehl interpretieren soll. Diese Mitteilung erfolgt durch die Reihenfolge, in der die 8-Bit-Datenworte dem Computer übermittelt werden (das erste Byte ist z.B. meist ein Befehl). Näheres hierzu siehe später bei der Erörterung des Befehlssatzes des Mikrocomputers.

Grundsätzlich kann also z.B. das hexadezimale Datenwort 47(H) folgendes für die Maschine bedeuten:

Datenwort: { 4 7 (H)
{ 0100 0111 (B) (nur dies "versteht" der Mikrocomputer)

Interpretation durch den Mikrocomputer
- als Zahl: 71 (D)
- als ASCII-Zeichen: G
- als Befehl: MOV B,A

Nach diesem grundsätzlichen Kapitel über die Darstellung von Information in einem binär arbeitenden System werden wir uns nun dem Aufbau und der Funktionsweise von Mikrocomputern zuwenden.

2) EINFÜHRUNG IN DIE MIKROCOMPUTER

2.1) Überblick über ein Mikrocomputer-System

Übernimmt ein Mikrocomputer die Steueraufgaben in einer komplexen digital-elektronischen Anlage (Apparatur) - einem elektronischen System - so nennt man das System ein Mikrocomputersystem (Abb.1).

Ein Mikrocomputer-System ist gekennzeichnet durch das Zusammenwirken von Mikrocomputer und Peripherie, wobei der Mikrocomputer das steuernde 'Gehirn' für den logischen Operationsablauf im System ist. Die Peripherie ist die zu steuernde Apparatur mit ihren Meßwertaufnehmern ("Sensoren"), Wandlern, Reglern, Schaltern usw., oder es sind konventionelle EDV-Peripherie-Geräte vorhanden, wie Drucker, Schreiber ("Plotter"), Magnetband- und Magnetplattengeräte, Datensichtgeräte ("Terminals") usw.

Der Mikrocomputer selbst besteht aus Zentraleinheit (CPU: Central Processor Unit), Speicher und Eingabe/Ausgabe-Einheit.

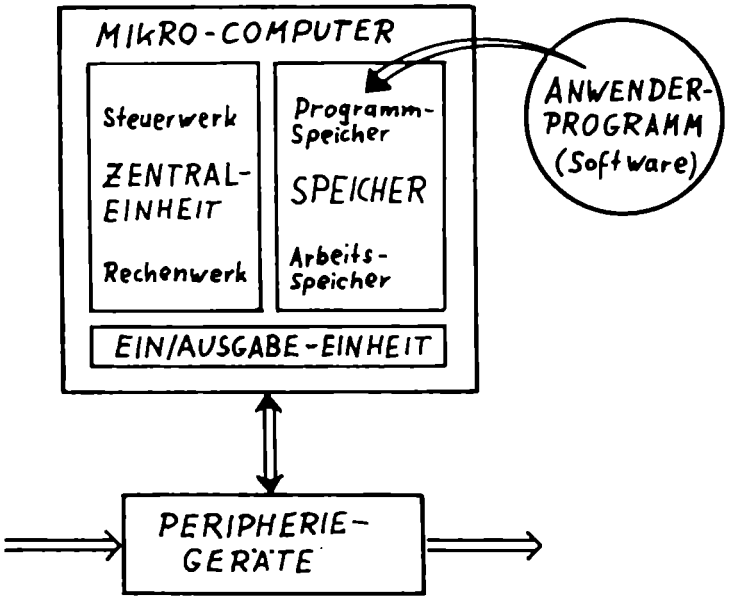


Abb. 1: Aufbau eines Mikrocomputer-Systems

Zentraleinheit (Zentralprozessor, CPU): Der wichtigste Bestandteil eines Mikrocomputers ist die CPU. Ihre Aufgabe ist es, die Verarbeitung der Daten durchzuführen und den Funktionsablauf im Computer zu steuern. Genauer gesagt: Die CPU holt Befehle aus dem Speicher und führt sie aus. Die Gesamtheit der nacheinander auszuführenden Befehle zur Erfüllung einer bestimmten Aufgabe heißt Programm. Die Zeit, die benötigt wird, um einen Befehl einzuholen und auszuführen, heißt Befehlszyklus; sie ist je nach Befehlsart verschieden lang. Damit setzt sich der zeitliche Ablauf eines Programmes aus einer Folge von kürzeren oder längeren Befehlszyklen zusammen.

Die Zentraleinheit besteht im wesentlichen aus Rechenwerk und Steuerwerk. Im Rechenwerk (ALU = Arithmetic Logic Unit) werden die arithmetischen und logischen Operationen durchgeführt. Das Steuerwerk sorgt für die Koordination des Mikrocomputers bei der Ausführung der Befehle.
Ist die Zentraleinheit eines Mikrocomputers als separater Baustein ausgeführt, so heißt sie Mikroprozessor.

Speicher: Der Speicher hat beim Mikrocomputer zwei Funktionen:

1.) Er enthält das Programm für die Durchführung einer bestimmten Aufgabe (Programmspeicher), d.h. die Folge der Befehls-Bitmuster in nicht löschbarer Form, sowie die Konstanten.

2.) Er dient der Speicherung der momentan "in Arbeit" befindlichen Daten (Arbeitsspeicher), sowie der Speicherung von Variablen.

Speicher sind elektronische Schaltungen zur Aufbewahrung von Bitmustern. Die Zahl der Speicherplätze nennt man "die Kapazität" des Speichers, z.B. kann der Speicher des Mikrocomputers 8080 auf eine Maximalkapazität von 64 K Byte ausgebaut werden, das sind 65536 Byte = 524 288 Bit.

Die Adresse eines Speicherplatzes ist eine Zahl (Bitmuster), mit deren Hilfe man eindeutig den gewünschten Speicherplatz finden kann ("Hausnummer"). Beim Mikrocomputer 8080 haben die Speicher-

plätze 16-Bit-Adressen. Es wird jedoch als Inhalt eines Speicherplatzes nicht das einzelne Bit adressiert, sondern immer ganze Datenwörter - etwa Datenwörter mit 8 Bit (1 Byte) Länge ("8 Bit organisierter Speicher").

Beispiel: Es sei im Speicherplatz mit der dezimalen Adresse 12 156(D) das 1-Byte-Datenwort 1001 1101 gespeichert.

Die 16-Bit-Adresse des Speicherplatzes lautet:

2	F	7	C	(H)
0010	1111	0111	1100	(B)

Der Inhalt dieses Speicherplatzes ist das 8-Bit-Datenwort:

1001 1101 = 9D (H)

Unterscheide also zwischen 16-Bit-Adresse und 8-Bit-Inhalt eines Speicherplatzes!

Eingabe/Ausgabe-Einheit: Die peripheren Apparaturen sind nicht Bestandteil des Mikrocomputers. Um Daten zwischen Mikrocomputer und peripheren Geräten auszutauschen, sind Eingabe/Ausgabe-Bausteine (E/A-, I/O-Bausteine) im Mikrocomputer notwendig, über die der Datenfluß erfolgt. Periphere Geräte können auf sehr unterschiedliche Weise an den Mikrocomputer angeschlossen werden, entsprechend vielfältig ist das Angebot an E/A-Bausteinen. Die Verbindungsanschlüsse der E/A-Bausteine zur Peripherie heißen E/A-Kanäle (Port), ein Kanal kann aus mehreren parallelen Leitungen bestehen. Wieviele Kanäle für ein bestimmtes peripheres Gerät benötigt werden, hängt vom Einzelfall ab.

Oft sind zwischen Mikrocomputer und peripheren Geräten zusätzliche Vorrichtungen zur elektrischen Anpassung erforderlich. Eine solche elektronische Anpassungs-Schaltung heißt Interface.

Man bezeichnet als Hardware die elektronischen plus technischen Teile des Computers und der angeschlossenen Geräte.
Software sind die Programme, zugehörige Notizen und Daten.

2.2) Ausführungsformen von Mikroprozessoren

Die kommerziell angebotenen Mikroprozessoren charakterisiert man in erster Linie nach ihrer "Wortlänge". Unter Wortlänge eines Mikroprozessors oder Mikrocomputers versteht man diejenige Anzahl von Bits, welche die Zentraleinheit in ihrem Rechenwerk "auf einmal" parallel verarbeiten kann.

Als weiteres Klassifikationsmerkmal von Mikroprozessoren dient die Technologie, in welcher der Mikroprozessor hergestellt ist:

Feldeffekt-Transistoren (MOS)	PMOS	(P-Kanal-MOS), langsamer als NMOS, "erste Generation"
	NMOS	(N-Kanal-MOS)
	CMOS	(Complementary MOS), geringe Leistungsaufnahme
	HMOS	(High density short channel MOS)
Bipolare Transistoren (sehr schnell)	Schottky TTL	(Transistor-Transistor Logic)
	I^2L	(Integrated Injection Logic)
	ECL	(Emitter Coupled Logic)

Heute sind im wesentlichen folgende Typen von Mikroprozessoren auf dem Markt:

- 4-Bit - Mikroprozessoren
- 8-Bit - Mikroprozessoren
- 16-Bit - Mikroprozessoren
- 32-Bit - Mikroprozessoren
- "Bit - Slice" - Mikroprozessoren (bipolar) mit 2, 4, (8) Bit, die man durch Kaskadieren zu einem 8-, 16-, oder 32-Bit-Mikroprozessor zusammenschalten kann.

(Ausnahme: es gibt auch 12-Bit-Mikroprozessoren)

Den größten Marktanteil besitzen z.Zt. die 8-Bit-Mikroprozessoren, wobei ein Trend zu 16-Bit-Mikroprozessoren erkennbar ist.

Größere Wortlänge:

- steigert die Arbeits-Geschwindigkeit des Mikrocomputers, da größere Bitmengen parallel verarbeitet werden und durch Verwendung "mächtigerer" Befehle weniger Befehle notwendig sind.
- bedeutet größere Universalität des Mikroprozessors durch komfortableren Befehlssatz und größere Speicherkapazität.
- bedingt jedoch höhere Hardware-Kosten.

Man kann den verschiedenen Mikroprozessor-Typen nach dieser Einteilung grob folgende Eigenschaften zuordnen:

4-Bit-Mikroprozessoren:
Kleiner Befehlsvorrat, kleiner Speicherbereich adressierbar, unbequem zu programmieren, langsam, billig, Alternative zu festverdrahteter Logik bzw. Kundenschaltkreis, Einsatz in kleinen Kontrollsystemen, viele andere Spezialanwendungen.

8-Bit-Mikroprozessoren:
Vielseitiger Befehlsvorrat, mittelschnell (aber noch langsamer als ein Minicomputer), brauchen Mehrwortbefehle, oft schon Compiler für höhere Sprachen verfügbar (z.B. für BASIC, FORTRAN, PASCAL,ADA, PL/M), Alternative zu festverdrahteter Logik bzw. Kundenschaltkreisen, Einsatz in kleineren und mittleren Systemen der Meß-, Steuer- und Regeltechnik.

16-Bit-Mikroprozessoren:
Befehlsvorrat wie ein Minicomputer, schnell (teilweise schon so schnell wie Minicomputer in konventioneller Technik), teuer, sehr universell, Einsatz in komplexen elektronischen Systemen, Alternative zu unausgelasteten Minicomputern.

32-Bit-Mikroprozessoren: Großrechner-Anwendungen

Mikroprozessoren in Bit-Slice-Architektur:
Zu beliebiger Wortlänge zusammensetzbare (kaskadierbare) Prozessoren, in bipolarer Ausführung sehr schnell, sehr aufwendige Programmierung, teuer, geeignet zur Emulation von Minicomputern (Emulation = softwaremäßige Nachbildung eines Computers).

Kriterien für die Beurteilung und Auswahl eines Mikroprozessors
Die nachfolgenden Kriterien sind für die Beurteilung eines Mikroprozessors von Bedeutung:
Datenformat (4, 8, 16, 32 Bit), Befehlsvorrat, Adressierungsmöglichkeit, CPU-Struktur, Mikroprogrammierbarkeit, Geschwindigkeit, Interrupt-Möglichkeiten, Schnittstelleneigenschaften, Speicherkonzept, Versorgungsspannung, Verfügbarkeit von Bausteinen, Zusatz-Hard- und Software für Systementwicklung, applikatorische Unterstützung des Anwenders durch den Hersteller, Preis.

2.3) Mikroprozessortypen und Mikroprozessorhersteller

(Diese Zusammenstellung enthält nicht die Ein-Chip-Mikrocomputer, deren Zahl stark zunimmt)

Typ	Wortlänge Technologie	Hersteller	Zweithersteller
4-Bit-Mikroprozessoren *)			
MCS-4004 -4040	4 Bit PMOS	Intel	
PPS-4	4 Bit PMOS	Rockwell	
8-Bit-Mikroprozessoren			
8080	8 Bit NMOS	Intel	Siemens,AMD,National Semiconduct.,Texas Instr., NEC,Mitsubishi,Toshiba, Hitachi,Signetics/Valvo
8085	8 Bit HMOS (CMOS)	Intel	Siemens,AMD,NEC,OKI, Mitsubishi,Toshiba
8088	8 Bit (16 Bit) HMOS	Intel	Siemens,Fujitsu,AMD
MC 6800/02/..	8 Bit HMOS (CMOS)	Motorola	AMI,Fairchild,Hitachi, Fujitsu,Thomson,Plessey
MC 6809	8 Bit (16 Bit) NMOS	Motorola	Thomson,AMI
6500/02/..	8 Bit NMOS (CMOS)	MOS-Technology /Commodore	Rockwell,Synertek
Z 80	8 Bit NMOS (CMOS)	Zilog	Mostek,Sharp,SGS-Ates, NEC,Hitachi
Z 800	8 Bit NMOS	Zilog	
8x300/05	8 Bit Schottky-TTL	Signetics/ Valvo	AMD
2650	8 Bit NMOS	Signetics/ Valvo	
8060 (SC/MP2) 8070	8 Bit NMOS	National Semiconductor	
NSC 800	8 Bit CMOS	National Semiconductor	
F8-3850	8 Bit NMOS	Fairchild	Mostek
CDP 1802/05/..	8 Bit CMOS	RCA	Hughes
9980/81	8 Bit (16 Bit) NMOS	Texas Instr.	AMI,ITT

*) Die neuen 4-Bit-Entwicklungen sind als komplette Ein-Chip-Mikrocomputer ausgelegt

Typ	Wortlänge Technologie	Hersteller	Zweithersteller
12-Bit-Mikroprozessoren			
IM 6100	12 Bit CMOS	Intersil	Harris Semicond.
16-Bit-Mikroprozessoren			
8086	16 Bit HMOS (CMOS)	Intel	Siemens,NEC,Harris, Fujitsu,Mitsubishi,AMD
80186	16 Bit HMOS	Intel	AMD
80286	16 Bit HMOS	Intel	Siemens,AMD
Z 8000/ 01/02/..	16 Bit NMOS	Zilog	AMD,SGS-Ates,Sharp
MC 68000/ 08/10	16 Bit (32 Bit) HMOS	Motorola	Rockwell,Thomson, Hitachi,Mostek, Signetics/Valvo
NS 16000/ 16/32	16 Bit (32 Bit) NMOS	National Se- miconductor	Fairchild,Synertek
TMS 9900/ 95	16 Bit NMOS	Texas Instruments	AMI,ITT
SBP 9900/ 89/95	16 Bit I2L	Texas Instruments	AMI,ITT
TMS 99000/ 105/110	16 Bit (32 Bit) NMOS	Texas Instruments	AMI
F 9440/9445	16 Bit I2L	Fairchild	
MN 601/602	16 Bit NMOS	Data General	
MICRO/J-11	16 Bit CMOS	Digital Equipment	
32-Bit-Mikroprozessoren			
iAPX 432	32 Bit HMOS	Intel	
80386	32 Bit CMOS	Intel	⎫
Z 80000	32 Bit NMOS	Zilog	⎬ Für 1984 angekündigt
MC 68020	32 Bit NMOS	Motorola	
NS 32032	32 Bit NMOS	National Semic.	⎭

Typ	Wortlänge Technologie	Hersteller	Zweithersteller
Bit-Slice-Prozessoren			
3000	2-Bit-Slice S-TTL	Intel	Signetics/Valvo
2900	4-Bit-Slice S-TTL	AMD	National Semicond., Motorola,Fairchild, Signetics etc.
9400	4-Bit-Slice S-TTL	Fairchild	Signetics/Valvo
MC 10800	4-Bit-Slice ECL	Motorola	
SBP 0400	4-Bit-Slice I2L	Texas Instr.	
SN 74 S 481	4-Bit-Slice S-TTL	Texas Instr.	

2.4) Zum Mikroprozessor 8080

Der Mikroprozessor 8080 der Fa. INTEL ist der direkte Nachfolger des ersten 8-Bit-Mikroprozessors der Welt, des 8008. Der INTEL 8008 wurde 1969 als Logiksystem für "intelligente Terminals" (Datensichtgeräte) entwickelt, kam jedoch dort nicht zum Einsatz und wurde von der Firma "aus Verlegenheit" 1971 als selbständiger, programmierbarer Logikbaustein auf dem Markt angeboten. Wenn man bedenkt, daß der 8008 zu einem Zeitpunkt entwickelt wurde, als noch gar keine bestimmten Mikrocomputer-Anwender existierten, ist die Genialität dieser Entwicklung frappierend.
Seit 1973 wird der INTEL 8080 als verbesserter Nachfolgetyp des 8008 auf dem Markt angeboten. Er wird von verschiedenen Firmen in Lizenz hergestellt und ist, obwohl schon ein "klassischer" Mikroprozessor, auch heute noch einer der bekanntesten 8-Bit-Mikroprozessoren. Er hat die weitere Entwicklung der 8-Bit-Mikroprozessoren entscheidend geprägt. So ist er beispielsweise der direkte Vorgänger der Mikroprozessoren 8085 und Z80, und wir werden sehen, daß einige Eigenheiten dieser beiden modernen Mikroprozessoren nur aus der historischen Entstehung aus dem Prozessor 8080 zu verstehen sind.

3) GROBSTRUKTUR DES MIKROCOMPUTERS

Wenn man den Aufbau eines Mikrocomputers nach seinen wichtigsten Funktions-Einheiten aufgliedert, kommt man zu dem für Mikrocomputer typischen Blockdiagramm von Abb. 2. Zentralprozessor (CPU), Speicherbausteine und Eingabe/Ausgabe-Bausteine sind bei größeren Mikrocomputersystemen getrennt hergestellte Bausteine, aus denen dann der Mikrocomputer erst vom Anwender nach Wunsch zusammengebaut wird ("Mehrchip-Mikrocomputer"), bzw. es werden vom Hersteller fertig verdrahtete Mehrchip-Systeme als "Einplatinen-Mikrocomputer" angeboten.
Es gibt jedoch auch komplette Mikrocomputer in Minimalausstattung auf einem einzigen Chip, bei denen sowohl der Zentralprozessor als auch die Speicher- und Ein/Ausgabe-Bausteine auf dem gleichen Chip voll integriert sind, sog. "Ein-Chip-Mikrocomputer" (teilweise mit getrenntem, extern zugeschaltetem ROM-Speicher). Diese Ein-Chip-Mikrocomputer stellen für einfachere Anwendungen sehr preisgünstige Problemlösungen dar.
Die Abb. 2 zeigt den funktionalen Aufbau eines Mikrocomputers, der auf dem 8-Bit-Mikroprozessor 8080 basiert:

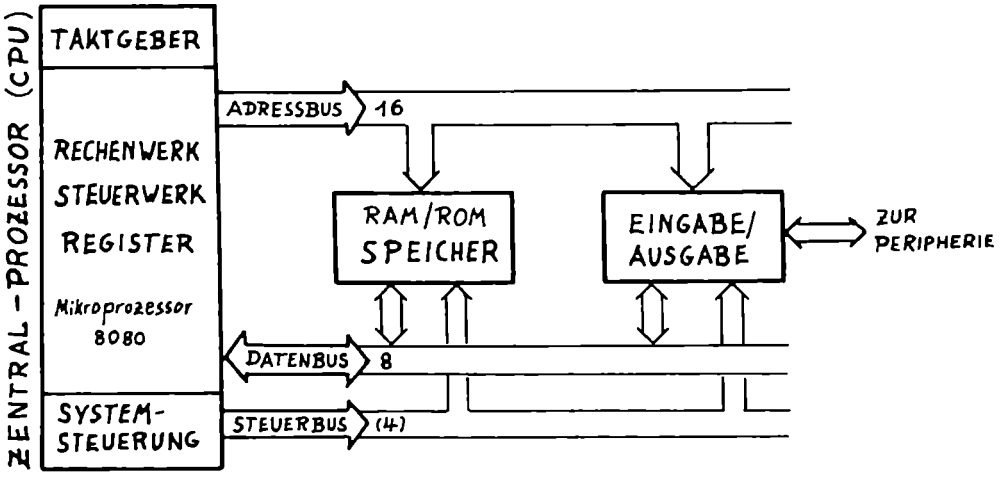


Abb. 2: Die wichtigsten Teile eines Mikrocomputers

3.1) BUS-Struktur

Charakteristisch für alle Mikrocomputer ist die Verbindung der verschiedenen Funktionsbausteine untereinander über "Bus"-Verbindungen.
Busse sind vieladrige, parallel verlaufende Leitungen, an welche die einzelnen Bausteine des Mikrocomputers gleichberechtigt angeschlossen sind. Über sie erfolgt der Austausch von Daten zwischen den einzelnen Bausteinen, werden Speicherplätze und Eingang/Ausgang-Kanäle adressiert und werden Steuersignale übermittelt. Man unterscheidet zwischen Daten-Bus, Adress-Bus und Steuer-Bus.
Durch die Bus-Struktur ist eine beliebige Erweiterung des Mikrocomputers durch zusätzliche Bausteine auf einfache Weise möglich.
Die Bus-Struktur wurde vom Minicomputer übernommen, Großcomputer arbeiten etwas anders (Multiplex-Kanäle, Selektorkanäle).

Im Mikrocomputer-System 8080 haben die 3 Busse folgende Gestalt:

- Daten-Bus: Er besteht aus 8 Leitungen, auf denen die Daten zwischen dem Zentralprozessor und den Speicher- oder E/A-Bausteinen in beiden Richtungen übertragen werden (bidirektional). Die 8 Leitungen entsprechen der 8-Bit-Wortlänge des 8080 Systems, so daß alle 8 Bit eines Datenwortes parallel (gleichzeitig) vom und zum Zentralprozessor übertragen werden können.

- Adress-Bus: Er besteht aus 16 Leitungen, über die ein bestimmter Speicherplatz oder ein bestimmter E/A-Kanal aufgerufen (adressiert) wird (Einweg-Datenübertragung). Die 16 Leitungen entsprechen den 16-Bit-Adressen des 8080 Systems, so daß alle 16 Adress-Bits parallel zum Speicher oder zur E/A-Einheit übertragen werden können.

- "Steuer-Bus": Er besteht aus 4 - 9 Leitungen, ist aber kein eigentlicher Bus. Es gibt nämlich keinen Steuerbus, an den alle Bausteine gleichberechtigt an-

geschlossen wären; vielmehr sind unterschiedliche Bausteine an unterschiedliche Steuer-Einzelleitungen angeschlossen.
Die Steuerleitungen sind notwendig, da infolge des Bus-Prinzips alle Einheiten des Computers gleichzeitig vom Bus mit Daten versorgt werden. Es muß dann dafür gesorgt werden, daß jeweils die richtige Einheit als Sender und die richtige Einheit als Empfänger aktiviert wird. Dies geschieht durch Steuersignale, die vom Zentralprozessor ausgehen.

3.2) Zentraleinheit (Zentralprozessor, CPU)

Die Zentraleinheit des Mikrocomputers besteht aus Rechenwerk, Steuerwerk, einer kleineren Anzahl von internen Registern, einer Systemsteuerungs-Einheit sowie einem Taktgenerator. Diese Funktionsgruppen sollen im folgenden für das Mikrocomputer-System 8080 beschrieben werden.

Taktgenerator (Clock)

Wie alle komplexeren digitalen Systeme arbeitet auch ein Mikrocomputer synchron und in zyklischem Zeitablauf. Das heißt, die logischen Operationen des Systems werden zu bestimmten Zeitpunkten vollzogen. Die hierzu benötigten Zeitmarken werden von einem zentralen Taktgenerator geliefert. Die Frequenz des Taktsignales ist ein wesentlicher Anhaltspunkt für die Arbeitsgeschwindigkeit eines Mikrocomputers. So beträgt beispielsweise für den Mikroprozessor 8080 A die schnellste Taktfrequenz 2 MHz.

Rechenwerk (ALU)

In der Zentraleinheit erfolgt die gesamte Verarbeitung der binären Daten, die im Laufe der Durchführung eines Programms vorgenommen werden muß. Diese Verarbeitung von Daten innerhalb der CPU wird hardwaremäßig in einer Gruppe von Digitalschaltungen

durchgeführt, die man als Rechenwerk (ALU = Arithmetic Logic Unit) bezeichnet. Im Rechenwerk werden ganze Bit-Gruppen parallel verarbeitet: Datenworte mit einer für den Mikroprozessor charakteristischen Länge. Im Falle des Mikroprozessors 8080 sind dies 8-Bit-Worte, d.h. der Mikroprozessor 8080 ist "8-Bit-organisiert".

Das Rechenwerk enthält hardwaremäßige Schaltungen zur Durchführung folgender 8-Bit-Operationen für Datenworte:

Arithmetische Operationen
Logische (Boole'sche) Operationen
Schiebe-Operationen

Register

Für die Zwischenspeicherung von Daten während eines Programmablaufes stehen intern im Zentralprozessor einige kleine Speichereinheiten zur Verfügung, sog. Register, mit 8 bzw. 16 Bit Speicherkapazität, in denen einzelne Datenworte abgelegt werden können. Da diese Register sich innerhalb der CPU befinden, kann man Zwischenergebnisse dort schneller speichern und wieder abrufen als im externen Speicher.

Steuerwerk (Control Unit)

Das Steuerwerk ist der komplizierteste Teil des Zentralprozessors. Es regelt und steuert den gesamten logischen Ablauf der Operationen, die bei der Ausführung eines Befehls innerhalb der CPU notwendig sind. Es liefert ferner (in Zusammenarbeit mit der Systemsteuerungs-Einheit) die Steuersignale für die Koordination der Mikrocomputer-Bausteine außerhalb des Zentralprozessors.

Die Bearbeitung eines Befehls durch den Zentralprozessor kann in 2 Teile zerlegt werden:

- in den einfacheren Vorgang des Befehls-Abrufs (Instruction Fetch), wobei das Steuerwerk veranlaßt, daß der Befehl aus dem Programmspeicher in ein internes Register der CPU geholt wird,
- und in den komplizierteren Teil der Befehls-Ausführung (Instruction Execution), wobei das Steuerwerk - je nach Art des

Befehls - eine Folge von Steuersignalen an die Logik-Bausteine der CPU abgibt, so daß der richtige Datenfluß und die zeitlich richtige Verknüpfung der Daten gewährleistet wird.

Oft muß das Steuerwerk auch auf externe Signale reagieren, z.B. auf eine Unterbrechungs- oder Warteanforderung. Eine Unterbrechungsanforderung durch ein peripheres Gerät veranlaßt die Steuerlogik, die Ausführung des Hauptprogramms zeitweilig zu unterbrechen und in ein Sonderprogramm zu springen. Eine Warteanforderung kann z.B. von einem Speicher- oder Eingabe/Ausgabe-Element erteilt werden, das langsamer als der Mikroprozessor arbeitet. Die Steuerlogik läßt dann den Mikroprozessor kurzzeitig stillstehen.(Siehe später.)

Systemsteuerung

In der mit Systemsteuerung bezeichneten Einheit werden die aus dem Steuerwerk kommenden Steuersignale aufbereitet, bevor sie die CPU endgültig über den "Steuerbus" verlassen und der Koordination der äußeren Mikrocomputer-Bausteine (Speicher, Eingabe/Ausgabe) dienen.

Mikroprozessor

Besteht ein Mikrocomputer aus einzelnen, separaten Bausteinen, so bezeichnet man mit "Mikroprozessor" denjenigen Baustein, der die Zentraleinheit des Mikrocomputers darstellt:

Mikroprozessor $\approx$ Zentralprozessor (CPU)

Der Mikroprozessor 8080 umfaßt jedoch nur Rechenwerk, Steuerwerk und Register, während Taktgenerator und Systemsteuerung als getrennte Bausteine vorliegen und erst zur vollständigen CPU zusammengeschaltet werden müssen. (Ein Grund ist der Engpaß der 40 Anschlußstellen (Pins) des Mikroprozessorgehäuses).

Beim Mikroprozessor 8085 ist dagegen die vollständige CPU auf einem Chip integriert. (Trick: Die Hälfte der Adreß-Bits wird im Zeit-Multiplex-Verfahren auf dem Datenbus übertragen, so daß man Pins für andere Zwecke zur Verfügung hat. Siehe später.)

3.3) Speicher

Bei einem 8-Bit-parallelen Mikrocomputer (wie z.B. dem System 8080) ist auch der Speicher 8-Bit-organisiert, d.h. die kleinste adressierbare Speichereinheit ist eine 8-Bit-Gruppe. Das hierin gespeicherte 8-Bit-Datenwort wird nach Aufruf parallel auf den 8 Datenleitungen zum Zentralprozessor übertragen, bzw. es wird ein Datenwort aus dem Zentralprozessor an die adressierte Stelle im Speicher gebracht und dort deponiert.
Der Speicher des Mikrocomputers hat - wie beschrieben - zwei Funktionen und hierfür auch zwei verschiedene Bauarten von Speichern (dies ist typisch für Mikrocomputer):

- Festwertspeicher: Zur endgültigen Speicherung von Programmen und Konstanten in nicht löschbarer Form (Programmspeicher). Der Inhalt dieses Speichers wird beim Abschalten der Versorgungsspannung nicht gelöscht; der Speicherinhalt kann auch nicht irrtümlich geändert werden. Man kann also aus diesem Speicher nur lesen, aber nicht in ihn hineinschreiben. Ein solcher Festwertspeicher heißt daher ROM (Read Only Memory).

- Arbeitsspeicher: Zur Speicherung von Zwischenergebnissen und Variablen und zur vorübergehenden Speicherung von Programmen (Programmentwicklung). In diesen Arbeitsspeicher müssen neue Daten (Bitmuster) immer wieder neu eingeschrieben werden können, d.h. der Speicherinhalt muß laufend geändert werden können. Man kann bei diesem Speicher also lesen und schreiben. Ein solcher Schreib-Lese-Speicher heißt RAM (Random Access Memory = Speicher mit wahlfreiem ("zufälligem") Zugriff). Im Gegensatz zum ROM bleiben in einem RAM die Daten bei Ausfall der Versorgungsspannung nicht erhalten. (Man behilft sich, wenn nötig, mit Pufferbatterien, um Stromausfälle zu überbrücken).

Ausführungsformen:
Für die Realisierung der beiden verschiedenen Typen von Mikrocomputer-Speichern (Festwertspeicher und Schreib-Lese-Speicher) wird heute hauptsächlich die MOS-Technologie verwendet[+)]. Von den Herstellerfirmen wird eine ganze Palette von Speicherbausteinen verschiedener Art und Größe angeboten:

[+)] Für extrem schnelle Speicher: bipolare Technologie

Festwertspeicher: Beim gewöhnlichen "maskenprogrammierten" ROM werden die Bitmuster (das Programm) dem Speicherbaustein beim Herstellungsprozeß über die Maske nach Kundenwunsch fest eingegeben. Der Speicherinhalt ist also später nicht mehr abänderbar. Wegen der kostenintensiven Maskenprogrammierung (jedes Programm erfordert eine eigene Maske) ist eine Anwendung erst bei größeren Stückzahlen (etwa ab 1000 Stück) üblich+).
Programmierbare Festwertspeicher: Das PROM (Programmable ROM) ist ein ROM, das man mit Hilfe eines Programmiergeräts selbst elektrisch programmieren kann. Das Programmieren erfolgt meist durch gezieltes Durchbrennen von Leitungsbrücken oder Dioden. Änderungen sind nur insoweit möglich, als man nachträglich bisher nicht programmierte Bits "einbrennt". Vorteil des etwas teureren PROM: Der Anwender kann die Programmierung selbst vornehmen.
Umprogrammierbare Festwertspeicher: Das EPROM (Erasable PROM) kann durch ein Quarzfenster mit Hilfe von ultraviolettem Licht "en bloc" gelöscht und mit einem Programmiergerät beliebig oft elektrisch neu programmiert werden. Dadurch ergibt sich die Möglichkeit der Fehlerkorrektur und der Programmänderung.
Wird auch der Löschvorgang elektrisch durchgeführt, so spricht man von einem EEPROM (E^2PROM, Electrically Erasable PROM); EEPROM's sind byteweise lösch- und programmierbar!

Schreib-Lese-Speicher: Bei den RAMs unterscheidet man zwischen statischen und dynamischen Bausteinen.
Bei einem statischen RAM (SRAM) bestehen die Speicherzellen aus Flipflops, in denen die Information so lange gespeichert bleibt, bis die betreffende Flipflopzelle adressiert und neu beschrieben wird. Bei einem dynamischen RAM (DRAM) werden zur Informationsspeicherung anstelle von Flipflops interne Halbleiter-Kapazitäten benutzt. Da deren Leckströme nicht unendlich klein sind, muß der Ladungsverlust in Abständen von einigen Millisekunden ausgeglichen werden. Dieser Vorgang heißt Auffrischen (Refresh), und man benötigt für ihn zusätzliche Hardware. Trotzdem lohnt sich das Verwenden dynamischer RAMs bei größerem Speicherbedarf, da ihre Speicherzellen kleiner sind als die der statischen RAMs und daher der Integrationsgrad je Chip höher gemacht werden kann.

+) In Festwertspeichern untergebrachte Software heißt auch "Firmware".

Die zur Zeit von den Herstellerfirmen angebotene maximale Speichergröße pro Baustein beträgt für

- maskenprogr.ROM : 1 M Bit
- EPROM : 256 K Bit
- PROM : 64 K Bit
- dynamisches RAM : 256 K Bit
- statisches RAM : 64 K Bit

Adressierung und Aufteilung des Speichers

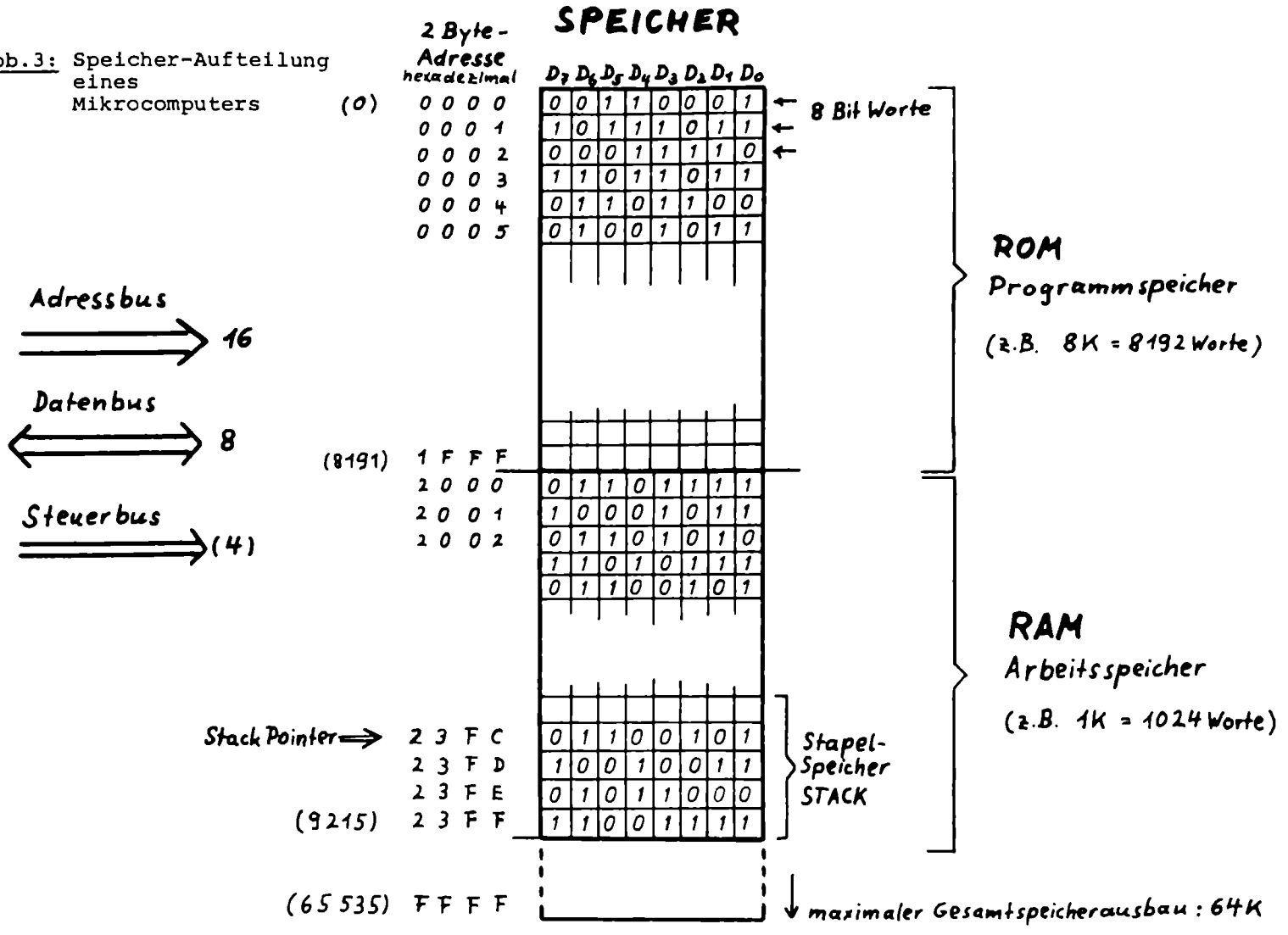
Der Gesamtspeicher eines Mikrocomputers ist bis zu einer typischen Maximalgröße ausbaubar. Diese Maximal-Anzahl der Speicherplätze wird durch die Länge (Bitzahl) der Adresse bestimmt. Beim System 8080 besitzt der Adreßbus 16 Einzelleitungen und entsprechend ist die Adresse 16 Bit lang. Es gibt daher 2^{16} = 65 536 = 64K verschiedene Adressen. Dies bestimmt die Maximalzahl der möglichen Speicherplätze. Der einzelne Speicherplatz ist dabei 8 Bit breit. (Will man also z.B. im Speicher eine Adresse (16 Bit) speichern, so benötigt man hierfür 2 Speicherplätze!)
Die möglichen Adressen laufen

von 0000 0000 0000 0000 = 0000(H) = 0(D)
bis 1111 1111 1111 1111 = FFFF(H) = 65535(D).

Von der Adressierung her besteht bei einem Mikrocomputer grundsätzlich kein Unterschied zwischen RAM und ROM. Der Anwender legt die Aufteilung des Gesamtspeichers je nach dem Bedarf an RAM-Plätzen (Arbeitsspeicher) und ROM-Plätzen (Programmspeicher) fest. Die Aufteilung der Adressen zwischen RAM und ROM hängt von der Größe der verwendeten Speicherbausteine ab.
Gewöhnlich teilt man dem ROM die niedrigsten Adressen zu. Der Grund hierfür ist der, daß nach dem Einschalten bzw. nach dem Rückstellen (Reset) der Zentralprozessor immer zuerst den Inhalt der niedrigsten Adresse einholt.
Vorübergehend kann ein Programm auch im RAM gespeichert werden (z.B. bei der Programmentwicklung). Es ist dann aber flüchtig (Achtung!).

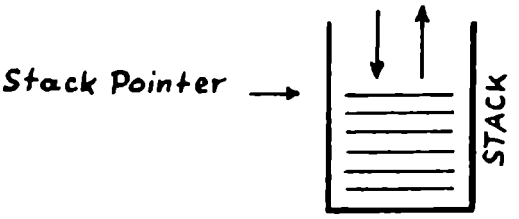
Ein Beispiel für eine mögliche Einteilung eines Speichers wird in Abb. 3 gegeben:

Abb.3: Speicher-Aufteilung eines Mikrocomputers



Der abgebildete Speicher besteht aus einem 8K Byte ROM-Speicher mit den Adressen 0000 bis 1FFF(H) und einem 1K Byte RAM-Speicher mit den Adressen 2000 bis 23FF(H).
Die 8-Bit-Worte in den einzelnen Speicherplätzen werden von rechts nach links mit den Nummern D_0- D_7 bezeichnet. Man nennt Bit 0 : LSB (Least Significant Bit), Bit 7 : MSB (Most Significant Bit).
Ebenso erfolgt die Kennzeichnung der Bits der 16-Bit-Adresse von rechts nach links mit A_0- A_{15}.

STACK: Ein Teil des RAM-Speichers (häufig die letzten Speicherplätze, d.h. diejenigen mit den höchsten Adressen) wird vom Anwender für einen Sonderspeicher reserviert: "Stapelspeicher" oder "Kellerspeicher": Stack. Wie wir sehen werden, wird durch ihn das Anspringen von Unterprogrammen (Routinen) und das anschließende Rückkehren in das Hauptprogramm auf einfache Weise möglich. Im Stack wird dabei diejenige Befehlsadresse niedergelegt, bei der das unterbrochene Programm später fortgesetzt werden soll. (Für diese Adressenspeicherung im Stack sind also jeweils 2 Speicherplätze erforderlich). Auch Mehrfachunterbrechungen, d.h. Programm-Verschachtelungen, sind mit Hilfe des Stapelspeichers möglich.
Im Unterschied zum normalen RAM-Speicher, in dem jede Adresse frei zugänglich ist, wird der Stapelspeicher geladen und entladen wie eine Groschenbox:



Man kann neue Daten nur "oben" einschreiben und auch nur die "oberste" Adresse abrufen:
LIFO-Betrieb: Last in-first out.
(Daher der Name "Stapelspeicher").
Damit bleibt bei Verschachtelungen des Programms die Reihenfolge erhalten.
Der Anwender legt fest, wieviele RAM-Plätze für den Stack reserviert bleiben sollen. Üblicherweise wird als erster zu belegender Stack-Speicherplatz die höchste RAM-Adresse (der "unterste" Platz in der Box) verwendet.

Der Zentralprozessor registriert in einem speziellen internen 16-Bit-Register, welche 16-Bit-Adresse im Stapelspeicher als letzte belegt worden ist. Dieses Register in der CPU heißt Stack-Pointer: Stapelzeiger. Der Stapelzeiger "zeigt" also immer auf die zuletzt belegte Stelle in der Stack-Box.

3.4) Eingabe/Ausgabe-Einheit (E/A bzw. I/O)

Die E/A-Bausteine ermöglichen den Datenaustausch zwischen dem Mikrocomputer und dem peripheren System (z.B. der zu steuernden Apparatur). Zwischen Mikrocomputer und externen Geräten werden Daten in beiden Richtungen über E/A-Kanäle (I/O-ports) übertragen.

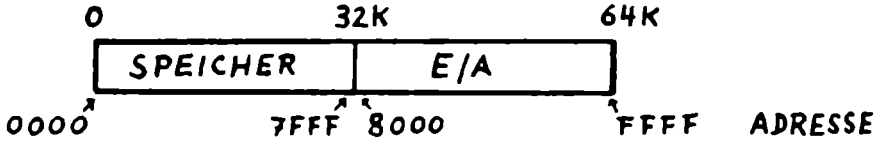
Bei einem 8-Bit-Mikrocomputer-System werden bei parallelem Datenaustausch jeweils 8 Bit je Kanal parallel übertragen, d.h. ein paralleler Kanal besteht in der Regel aus 8 Leitungen.
Zusätzlich stehen E/A-Bausteine zur Verfügung, welche eine serielle Datenübertragung nach außen ermöglichen, d.h. die aus dem Zentralprozessor kommenden parallelen Daten werden nach Zwischenspeicherung im E/A-Baustein in einen seriellen Datenstrom umgewandelt, der den Mikrocomputer nach außen verläßt. Umgekehrt werden im seriellen E/A-Baustein von außen ankommende serielle Daten in 8-Bit-parallele Daten für den Mikrocomputer umgewandelt.

Die Adressierung der Kanäle erfolgt über "Kanalnummern". Sie entsprechen den Adressen bei den Speicherplätzen und werden auf dem Adressbus vom Zentralprozessor ausgesandt. Man unterscheidet 2 Möglichkeiten der Adressierung:

a) Isolierte Ein/Ausgabe (Isolated I/O): Standard-Adressierung. Die CPU unterscheidet hierbei durch getrennte Steuersignale zwischen Speicher- und Eingabe/Ausgabe-Einheit: Wenn der Zentralprozessor einen Befehl erhalten hat, der den Datenaustausch mit einem E/A-Baustein anordnet, dann sendet er Steuersignale aus, die den Speicher inaktiv lassen, dagegen die E/A-Kanäle aktivieren; das Bitmuster auf dem Adressbus wird dann als Kanalnummer wirksam, nicht als Speicheradresse.
In mnemonischer Schreibweise lautet beim Mikroprozessor 8080
der Eingabebefehl: IN nr
der Ausgabebefehl: OUT nr (dabei ist "nr" die Kanalnummer)

Im Gegensatz zur Speicher-Adresse ist die Kanalnummer nur 8 Bit lang - es wird nur das niedrigere Byte der Adresse verwendet. Man hat also die Möglichkeit, bei dieser Adressierungsart maximal 2^8 = 256 verschiedene Kanäle aufzurufen. (Diese Anzahl genügt in den meisten Fällen).

b) Speicher-Ein/Ausgabe (Memory-Mapped-I/O): Bei dieser speziellen Adressierungsart werden die E/A-Kanäle als Speicherplätze aufgefaßt und mit einer 16-Bit-Adresse adressiert, wobei man einen Teil der Adressen für die E/A-Kanäle reserviert.
Natürlich wird dadurch die Gesamtzahl der maximal verfügbaren Speicherplätze um die Zahl der Kanäle reduziert, aber die volle Speicherkapazität von 65 536 Byte wird in den seltensten Fällen beim Einsatz von Mikrocomputern für die Steuerung von digitalen Systemen benötigt.
Braucht man z.B. nicht mehr als die Hälfte der möglichen Speicherplätze, so kann man die erste Hälfte der Adressen den Speicherplätzen zuordnen und die andere Hälfte der Adressen den E/A-Kanälen zuweisen: Es werden also in diesem Falle alle mit 0 beginnenden Adressen zur Adressierung von Speicherplätzen verwendet, alle mit 1 beginnenden Adressen werden zum Aufruf von E/A-Kanälen verwendet.



Der Vorteil dieser Adressierungsart ist, daß die E/A-Kanäle softwaremäßig gleich behandelt werden wie die Speicherplätze (gleiche Befehle), was elegantere Programme ermöglicht.
Voraussetzung ist dabei aber, daß sich die an die Kanäle angeschlossenen Geräte auch physikalisch wie Speicherplätze benehmen. Dies ist nicht der Fall bei langsameren Geräten. Diese müssen Warte-Anforderungen an den Mikrocomputer stellen, benehmen sich also nicht wie schnelle Speicherplätze!

4) BEFEHLSABLAUF IM MIKROCOMPUTER

4.1) Struktur der Befehle

Ein Anwender-Programm besteht aus einer längeren Reihe von Befehlen, welche als Bitmuster von je 8 Bits im Programmspeicher (z.B. in einem nichtflüchtigen ROM) abgelegt sind. Jeder Befehl löst eine elementare Operation aus, z.B. eine Datenübertragung, eine arithmetische oder logische Operation mit einem Datenbyte usw.

Die Befehle sind entsprechend der logischen Reihenfolge des Programms im Speicher in aufeinanderfolgenden Speicherplätzen abgelegt. Das Programm läuft normalerweise sequentiell in der Reihenfolge der aufeinanderfolgenden Befehlsadressen ab, beginnend bei der Anfangsadresse. Abweichungen hiervon sind durch verschiedene Arten von Sprungbefehlen möglich, mit deren Hilfe Programmsprünge an irgend eine Stelle des Programms erfolgen können.

Bei einem 8-Bit-Mikroprozessor mit einer Befehls-Wortlänge von 8 Bit sind 256 verschiedene Befehle möglich, beim Mikroprozessor 8080 werden hiervon 244 ausgenutzt. Will man mehr Information in einen Befehl packen, so muß man auf Mehrwort-Befehle ausweichen. Der Mikroprozessor 8080 kennt 1- bis 3-Wort-Befehle mit 1 bis 3 Byte Länge.

Das 1. Byte eines Befehles stellt immer den Hauptteil des Befehls dar, es legt den allgemeinen Charakter des Befehls fest. Dieses 1. Befehlsbyte heißt Operationscode (Opcode). Viele Befehle bestehen ausschließlich aus dem Operationscode. Für bestimmte Befehle kommt ein 2. oder 3. Byte hinzu zur Kennzeichnung von Daten oder Adressen, auf die der Befehl angewendet werden soll. Diese zusätzlichen Bytes heißen Operand.
Es gibt beim Mikroprozessor 8080 also 1-, 2- und 3-Byte-Befehle, sie belegen entsprechend 1,2,3 aufeinanderfolgende Speicherplätze:

1-Byte-Befehl: Nur Operationscode
2-Byte-Befehl: Operationscode + Operand (8-Bit-Kanalnummer oder 8-Bit-Datenwort)
3-Byte-Befehl: Operationscode + Operand (16-Bit-Adresse oder 16-Bit-Datenwort)

Beispiel für ein kleines Programm:
========
Es sei der 1. Befehl: "Bringe den Inhalt des Registers A der Zentraleinheit ins Register B".
Hierbei handelt es sich für den Mikroprozessor 8080 um einen 1-Byte-Befehl, der nur aus dem Operationscode besteht:

Bitmuster des Befehls:	0100 0111	Maschinensprache
Hexadezimales Kürzel:	4 7	
Mnemonische Schreibweise:	MOV B,A ("move register A to register B")	ASSEMBLER-Sprache

Dieser uns schon bekannte Befehl soll der erste Befehl eines Programms sein, wir legen ihn im Speicher z.B. an den Platz Nr. 4, seine Adresse ist also 0000 0000 0000 0100.

Es sei der 2. Befehl: "Lade das (freigewordene) Register A mit dem Inhalt einer bestimmten Adresse des Speichers", z.B. dem Inhalt der Adresse 0110 1010 0010 1101 (Hexadezimal: 6A2D).
Dies ist ein 3-Byte-Befehl, neben dem Operationscode ist zur Kennzeichnung der Adresse noch ein 2-Byte-Operand erforderlich:

Bitmuster des Befehls:	0011 1010	Opcode (hexadezimal 3A)
	0010 1101	Operand niederes Byte (2D)
	0110 1010	Operand höheres Byte (6A)

Mnemonische Schreibweise: LDA 6A2D H
(" Load register A direct")

Diese 3 Bytes speichern wir an den 3 nächsten Speicherplätzen 5, 6 und 7, wobei die Vorschrift zu beachten ist, daß das höhere Byte des Operanden nach dem niederen Byte abgelegt wird[+].

[+] Dies entspricht dem 2-Byte-Befehl, wo auch das (niedere) Operandenbyte direkt hinter dem Opcode liegt.

Das kleine Programm unseres Beispiels würde also wie folgt im Programmspeicher abgespeichert sein:

	Speicher-Adresse	Speicher-Inhalt
(0)	0000 0000 0000 0000	-
(1)	0000 0000 0000 0001	-
(2)	0000 0000 0000 0010	-
(3)	0000 0000 0000 0011	-
(4)	0000 0000 0000 0100	0100 0111
(5)	0000 0000 0000 0101	0011 1010
(6)	0000 0000 0000 0110	0010 1101
(7)	0000 0000 0000 0111	0110 1010
(8)	0000 0000 0000 1000	-

4.2) Ablauf eines Befehlszyklus

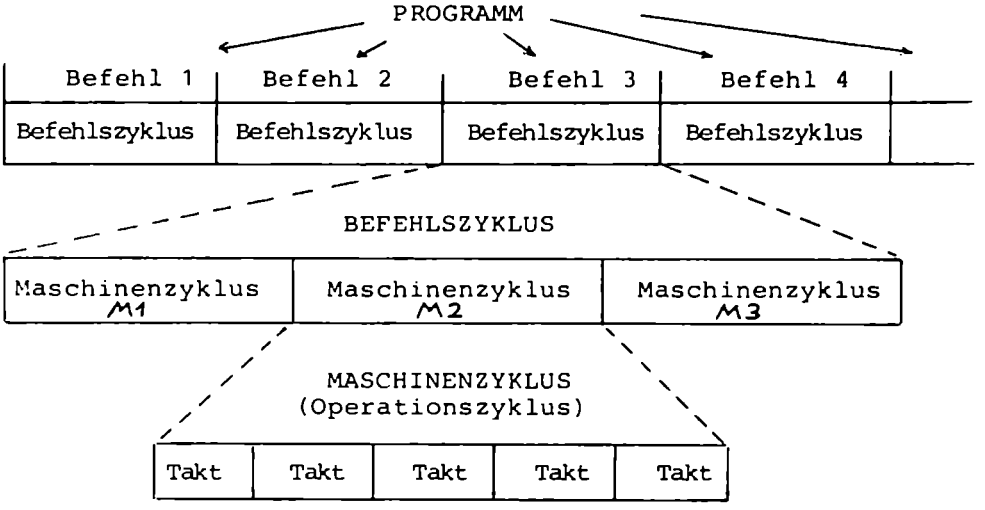
Der zeitliche Ablauf eines Befehles im Mikrocomputer kann in zwei Phasen zerlegt werden: den Befehlsabruf und die Befehlsausführung. Während des Abrufs wird der aus 1 - 3 Bytes bestehende Befehl aus dem Programm-Speicher in die CPU geholt und dort in Registern gespeichert. Während der Ausführungsphase wird der Befehl vom Steuerwerk der CPU dekodiert, es werden vom Steuerwerk Steuersignale zur Durchführung des Befehlsinhalts gebildet und an die entsprechenden Stellen im Mikrocomputer gesandt und so der Befehl ausgeführt.

4.21) Definition von Zeitspannen beim Befehlsablauf

Takt: Die Arbeit eines Computers wird zeitlich koordiniert durch ein Taktsignal. Dieses Signal zerlegt die Arbeitszeit in gleiche Takte. Die Länge eines Taktes (bzw. die Taktfrequenz) ist ein Maß für die Arbeitsgeschwindigkeit eines Mikroprozessors. So beträgt beispielsweise beim Mikroprozessor Typ 8080 A die kürzeste Taktlänge 0,48/us, was einer Taktfrequenz von etwa 2 MHz entspricht.[+)]

[+)] Beim Typ 8080 A-1: Kürzester Takt 0,32/us (entspricht 3 MHz)
Beim Typ 8080 A-2: Kürzester Takt 0,38/us (entspricht 2,6 MHz)

Befehlszyklus: Die gesamte Zeitspanne des Einholens und Ausführens eines Befehles wird als Befehlszyklus bezeichnet. Ein Befehlszyklus kann beim Mikroprozessor 8080 je nach Art des Befehles zwischen 4 und 18 Takte dauern, beim 0,5 µs-Takt also zwischen 2 µs und 9 µs. Man nennt gleichzeitig den kürzesten vorkommenden Zyklus "den Befehlszyklus des Mikroprozessors", beim Mikroprozessor Typ 8080 A beträgt er also 2 µs.+)



Maschinenzyklus (Operationszyklus): Eine anschauliche Vorstellung vom Arbeiten eines Mikrocomputers liefert die Unterteilung der Befehlszyklen in Maschinenzyklen. Vereinfacht gesprochen beginnt ein neuer Maschinenzyklus immer dann, wenn der Mikroprozessor in einen neuen Datenaustausch mit Speicher oder E/A-Einheit treten muß (wenn die CPU verlassen werden muß). Die Anzahl der Maschinenzyklen, die während eines Befehlszyklus vorkommen, hängt von der Art des Befehles ab, beim Mikroprozessor 8080 beträgt sie 1 - 5 Maschinenzyklen pro Befehlszyklus. Jedenfalls ist der erste Maschinenzyklus M1 eines Befehlszyklus immer ein Befehlsabruf. Die Zeitspanne für einen einzelnen Maschinenzyklus beträgt 3 bis 5 Takte.
Es gibt beim System 8080 zehn verschiedene Arten von Maschinenzyklen, die innerhalb eines Befehlszyklus auftreten können:

+) Beim Typ 8080 A-1: 1,3 µs
Beim Typ 8080 A-2: 1,5 µs

1. Befehlsabruf (FETCH): M1
2. Speicher lesen (MEMORY READ)
3. Speicher einschreiben (MEMORY WRITE)
4. Stapelspeicher lesen (STACK READ)
5. Stapelspeicher einschreiben (STACK WRITE)
6. Eingabe (INPUT)
7. Ausgabe (OUTPUT)
8. Unterbrechung (INTERRUPT)
9. Halt (HALT)
10. Unterbrechung während Halt (HALT - INTERRUPT)

(Der Prozessor 8080 kennzeichnet jeden Maschinenzyklus, indem er während des ersten Taktes des Maschinenzyklus kurzzeitig ein spezifisches 8-Bit-Zustandswort auf dem 8080-Datenbus an die Systemsteuerungseinheit aussendet (STATUS-Wort). Dieses Status-Wort wird zur Bildung der Steuersignale verwendet, welche gezielt die Speicherbausteine oder die Ein/Ausgabe-Bausteine aktivieren. Wir werden später auf das Status-Wort noch kurz eingehen.)

In der ausführlichen Darstellung der Befehle des Mikroprozessors 8080 im Kapitel 7 ist für jeden der 244 Befehle die für den Befehlszyklus benötigte Zahl von Maschinenzyklen und Takten angegeben.

4.22) Beispiele für Befehlszyklen

Als 1. Beispiel wird der Befehl MOV B,A von früher benutzt ("Bringe Inhalt von Register A ins Register B"). Als erste Maßnahme des Zentralprozessors wird über den Steuerbus den anderen Bausteinen mitgeteilt: "Speicher lesen und zwar Befehl einholen". Woher der Befehl geholt wird, das bestimmt die nun vom Zentralprozessor anzugebende Adresse. Diese Adresse findet der Mikroprozessor in einem internen "Befehlszähler" (Program Counter). Das ist ein 16-Bit-Register nur für diesen speziellen Zweck. Es wird vom Mikroprozessor selbst auf dem Laufenden gehalten: Immer wenn ein Byte eines Befehls eingeholt wird, inkrementiert der Mikroprozessor den Befehlszähler um 1 auf die nächste Befehlsadresse (der Befehlszähler kann durch Sprungbefehle

(s.später) auch auf andere Adressen gesetzt werden). Nun sendet der Mikroprozessor den Inhalt des Befehlszählers auf den Adressbus. Dies geschieht im 1. Takt des 1. Maschinenzyklus des Befehlszyklus.
Daraufhin gibt der durch die Steuerleitung $\overline{\text{MEMR}}$ (= Speicher lesen) aktivierte Speicher den Inhalt des adressierten Speicherplatzes auf den Datenbus. Dies geschieht im 3. Takt. Der Mikroprozessor "weiß" aus der Reihenfolge des Ablaufs, daß er dieses Byte als Operationscode zu interpretieren hat. Er legt es in einem speziellen internen 8-Bit-"Befehlsregister" ab. Zuvor hat der Mikroprozessor im 2. Takt den Befehlszähler um 1 inkrementiert und verfügt damit über die nächste Befehlsadresse (oder bei Mehr-Byte-Befehlen über die Adresse des 2. Bytes).
Die Ausführung des Befehls "47(H)"- d.h. MOV B,A - benötigt keinen weiteren Maschinenzyklus mehr, denn die Ausführung des Befehls erfordert nur eine Datenverschiebung im Innern des Mikroprozessors. Diese Befehlsausführung findet im 4. und 5. Takt des ersten (und einzigen) Maschinenzyklus statt. (Bezüglich der erwähnten internen Register des Mikroprozessors vergleiche auch Abb. 4 im Kapitel 5).

Anders ist es im 2. Beispiel bei dem uns auch schon bekannten Befehl LDA adr ("Lade Register A mit dem Inhalt eines Speicherplatzes"). Schon das Einholen dieses Befehles erfordert 3 Maschinenzyklen, da der Befehl ein 3-Byte-Befehl ist (neben dem Opcode 3A existiert eine 16-Bit-Adresse als Operand). Das 2. und 3. Byte des Befehls wird in internen Hilfsregistern W und Z im Mikroprozessor abgelegt. Bei der Ausführung des Befehles - Einholen des Speicherplatz-Inhaltes und Ablegen im Register A - muß die CPU verlassen werden, was einen weiteren Maschinenzyklus erfordert. In diesem 4. (und letzten) Maschinenzyklus sendet der Prozessor die gewünschte Speicheradresse auf dem Adressbus aus, aktiviert den Speicher durch die Steuerleitung $\overline{\text{MEMR}}$, holt den Inhalt des adressierten Speicherplatzes auf dem Datenbus herein und legt ihn im Register A ab.

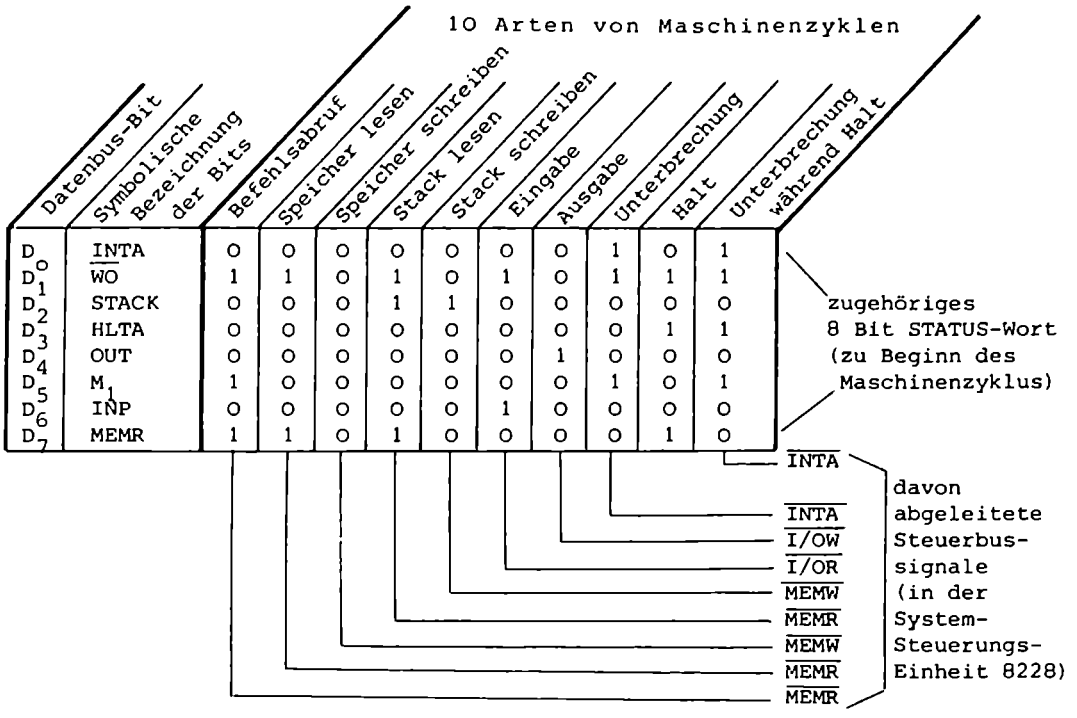
Schließlich sei noch der Befehl INR M erwähnt ("Inkrementiere einen Speicherplatz-Inhalt um 1"). Dies ist ein 1-Byte-Befehl (34(H)), der ganze Befehl ist also nach dem 1. Maschinenzyklus eingeholt. Aber die Ausführung braucht 2 weitere Maschinenzyklen; denn der Inhalt des Speicherplatzes M muß erst zur Inkrementierung in die CPU geholt, und das inkrementierte Byte muß dann wieder in den Speicher zurückgebracht werden.

Eingabe- und Ausgabe-Befehle sind in ihrem Ablauf ähnlich wie Lese- und Einschreib-Operationen für den Speicher, außer daß jetzt Eingabe/Ausgabe-Kanäle anstelle von Speicherplätzen adressiert werden. Der Mikroprozessor sendet ein entsprechendes Eingabe- oder Ausgabe-Steuersignal aus, das die E/A-Einheit aktiviert und adressiert den gewünschten E/A-Kanal. Daraufhin empfängt der Mikroprozessor entweder Eingabedaten, die von einem Peripherie-Gerät an die E/A-Einheit übertragen werden, oder der Mikroprozessor sendet Daten aus, die über die E/A-Einheit an ein Peripherie-Gerät gelangen.

4.23) Kennzeichnung der Maschinenzyklen durch STATUS-Worte +)

Wie bereits erwähnt, kennzeichnet der Mikroprozessor 8080 den Beginn eines neuen Maschinenzyklus dadurch, daß er während des 1. Taktes ein charakteristisches 8-Bit-Zustandswort (STATUS-Wort) an die Systemsteuerungseinheit aussendet. Dieses Statuswort erscheint kurzzeitig auf den Datenleitungen des 8080-Datenbusses und kennzeichnet jeden der 10 Maschinenzyklen auf eindeutige Weise. Das Statuswort wird in der Systemsteuerungs-Einheit des Zentralprozessors gespeichert und daraus die Steuersignale für den Steuerbus gebildet. Die folgende Tabelle enthält die Bitmuster der Status-Worte der 10 verschiedenen Maschinenzyklen. Die hiervon abgeleiteten Steuersignale sind rechts unten aufgeführt. Nur diese Steuersignale sind für den Anwender des 8080-Systems von Interesse.

+) Dieses Kapitel kann beim ersten Lesen übersprungen werden



Die einzelnen Bits der Statusworte haben symbolische Bezeichnungen. Diese symbolischen Bezeichnungen haben dabei folgende Bedeutung:

INTA = 1: bedeutet Quittungssignal für Unterbrechungs-Anforderung. (Es wird auch verwendet, um einen RESTART-Befehl auf den Datenbus zu leiten).

$\overline{WO}$ = 0: zeigt an, daß die Operation im gegenwärtigen Maschinenzyklus entweder eine "Speicher-Schreib"- oder eine "Ausgabe"-Operation ist. Andernfalls wird eine "Speicher-Lese"- oder eine "Eingabe"-Operation ausgeführt ($\overline{WO}$ = 1).

STACK = 1: zeigt an, daß auf dem Adreßbus die Adresse des Stapelzeigers liegt.

HLTA = 1: bedeutet Quittungs-Signal für HALT-Befehl.

OUT = 1: zeigt an, daß auf dem Adreßbus die Adresse eines Ausgabekanals liegt, und daß der Datenbus die Ausgabedaten enthalten wird, sobald WR aktiv wird.

M1 = 1: zeigt an, daß sich der Mikroprozessor 8080 in einem Abruf-Zyklus zum Einholen des ersten Befehls-Bytes (Operationscode) befindet.

INP = 1: zeigt an, daß auf dem Adreßbus die Adresse eines Eingabeelements liegt und die Eingabedaten auf den Datenbus gegeben werden.

MEMR = 1: zeigt an, daß der Datenbus Speicher-Lese-Daten aufnimmt.

5) BLOCKSCHALTBILD EINES STANDARD-SYSTEM-AUFBAUS EINES MIKROCOMPUTERS MIT DEM MIKROPROZESSOR 8080

Das Blockschaltbild von Abb. 4 zeigt den typischen Systemaufbau eines 8-Bit-Mikrocomputers unter Benutzung von handelsüblichen Standard-Bausteinen. Das System besteht aus der Zentraleinheit mit unserem exemplarischen Mikroprozessor 8080, der Speichereinheit mit verschiedenen ROM- und RAM-Bausteinen, sowie der Eingabe/Ausgabe-Einheit mit parallelen und seriellen E/A-Kanälen. Dieser Standard-Systemaufbau ist für viele Anwendungszwecke ausreichend.

5.1) Zentraleinheit (Zentralprozessor, CPU)

Die Zentraleinheit besteht beim 8080-System aus 3 Bausteinen: dem eigentlichen Mikroprozessor 8080, dem Taktgeber 8224 und einer Systemsteuerungs-Einheit 8228. Diese beim 8080-System noch getrennten Bausteine gehören funktionell zum Zentralprozessor zusammen. (Ihre Trennung ist einerseits historisch zu erklären, andererseits durch den Engpaß der 40 Gehäuse-Anschlüsse des Mikroprozessors mitbedingt). Wir werden daher den Zentralprozessor als eine Funktionseinheit betrachten und in dieser grundlegenden Betrachtung nur auf diejenigen Leitungen im Blockschaltbild genauer eingehen, die den Zentralprozessor verlassen.

5.11) Taktgeber 8224 (Taktgenerator und Leistungstreiber)

Der Mikroprozessor 8080 arbeitet synchron, er benötigt also zur zeitlichen Steuerung der Abläufe einen Taktgeber (clock generator). Außerdem ist der Mikroprozessor 8080 eine dynamische Schaltung, d.h. seine internen Speicherelemente und Logikschaltungen brauchen zum Wiederauffrischen geeignete Refresh-Impulse. Für seinen Betrieb benötigt der Mikroprozessor 8080 zwei verschiedene Taktsignale Ø1 und Ø2, die zueinander eine bestimmte Phasenbeziehung haben müssen (nicht überlappend). (Die modernen Mikroprozessoren 8085 und Z80 benötigen dagegen nur noch ein einfaches Taktsignal.)

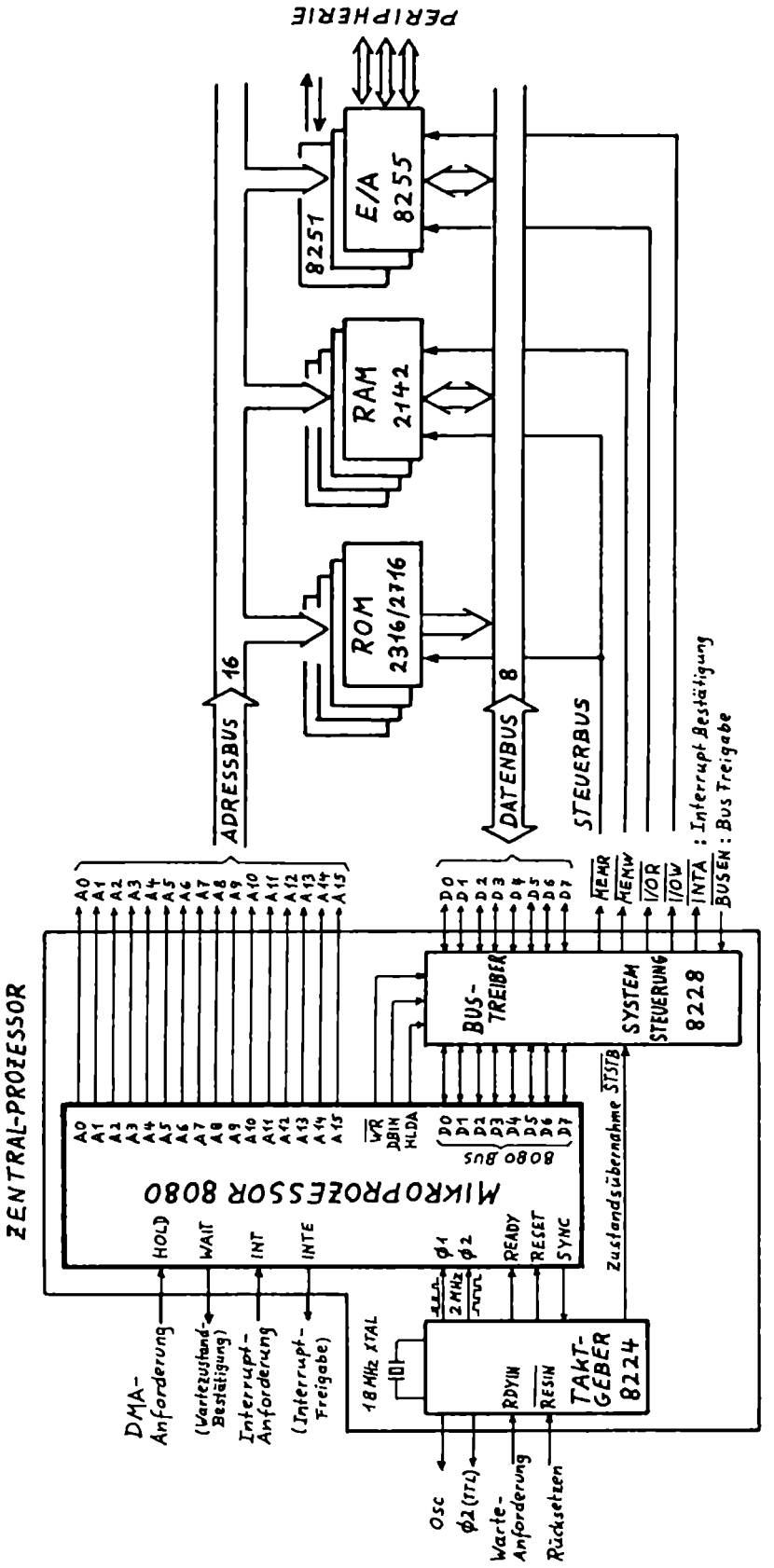
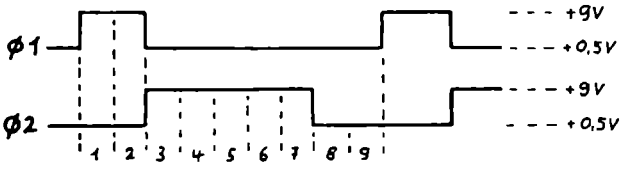


Abb.4: Mikrocomputer. Blockschaltbild für einen Standard System Aufbau mit dem Mikroprozessor 8080

Der Taktgeber besteht aus einem Quarzoszillator mit extern zuschaltbarem Quarz (mit einer Frequenz von z.B. 18 MHz für den Mikroprozessor 8080 A), ferner einer Impulsformerstufe. Die Impulsformerstufe macht über einen 1 : 9 Teiler und weitere logische Verknüpfungen die beiden 8080-Taktimpulse Ø1 und Ø2 (mit einer Taktfrequenz von 2 MHz beim 8080 A). Das Impulsdiagramm für diese beiden 8080-Taktsignale sieht folgendermaßen aus:



5.12) Mikroprozessor 8080+) : Rechenwerk, Steuerwerk, Register

Der Mikroprozessor 8080 kann funktionell gegliedert werden in: Rechenwerk (ALU), Steuerwerk und eine Reihe von 8- bzw. 16-Bit-Registern. Bei den Registern unterscheidet man zwischen:

- "universelle Register", die auch dem Operateur zur Verfügung stehen (A,B,C,D,E,H,L)
- Register für spezielle Aufgaben: Befehlszähler, Befehlsregister, Stapelzeiger, Bedingungsregister (Flag-Register)
- sowie weitere interne Register, die für den Funktionsablauf des Mikroprozessors benötigt werden und auf die der Operateur keinen Zugriff hat.

In den Blockschaltbildern von Abb. 5 ist der Aufbau des Mikroprozessors 8080 in Grobform und in Feinform dargestellt.

Rechenwerk (ALU)

Alle Mikroprozessoren enthalten eine Arithmetik-Logik-Einheit, die meistens abgekürzt als "ALU" bezeichnet wird. Die ALU ist derjenige Teil der Hardware im Mikroprozessor, der mit den Binärdaten arithmetische und logische Operationen durchführt.

+) Lieferbar in den Versionen 8080 A (2 MHz), 8080 A-2 (2,6 MHz) und 8080 A-1 (3 MHz), die sich in der maximalen Taktfrequenz unterscheiden.

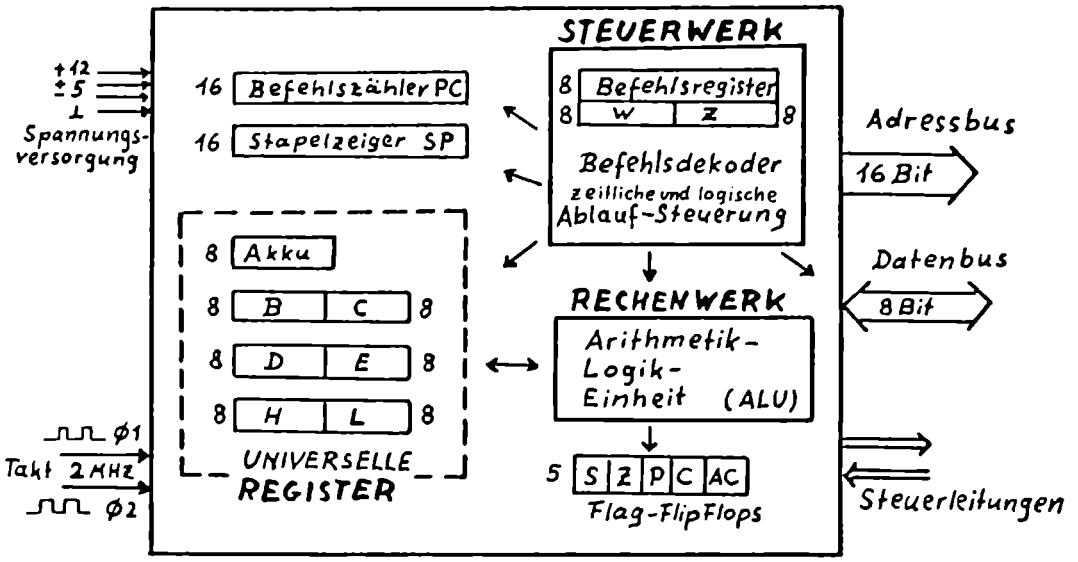
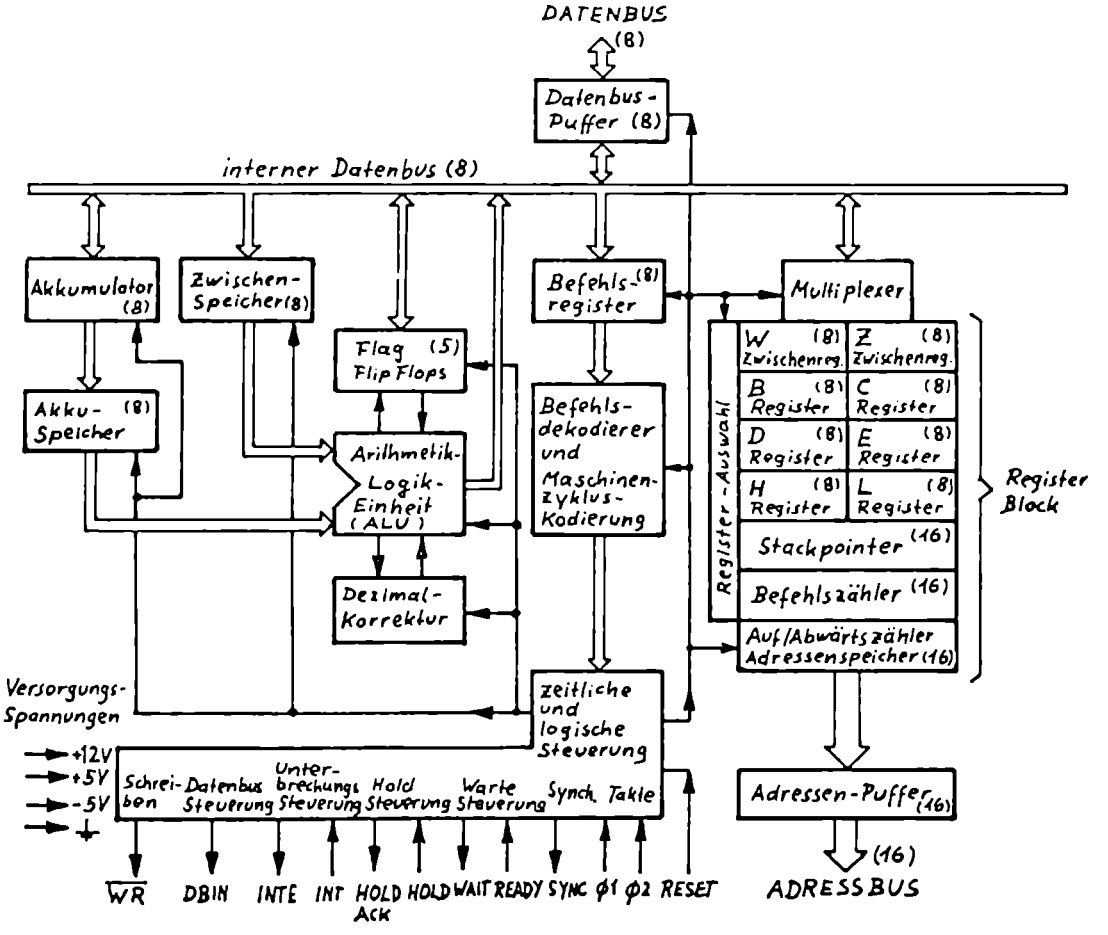


Abb.5: Innerer Aufbau des Mikroprozessors 8080 A
in Grobform (oben) und im verfeinerten Blockschaltbild (unten)



Der Mikroprozessor 8080 kann hardwaremäßig folgende 8-Bit-parallele Operationen mit 8-Bit-Datenworten (teilweise auch mit 16-Bit-Datenworten) durchführen:

- Arithmetische Operationen: Addieren, Subtrahieren (in Zweierkomplement-Arithmetik), Inkrementieren, Dekrementieren
- Boole'sche Logik-Operationen: UNDieren, ODERieren, Invertieren, EXCLUSIV-ODERieren
- Schiebe-Operationen: Ein 8-Bit-Muster um 1 Stelle nach links oder rechts verschieben.

Alle komplizierteren mathematischen Operationen müssen vom Anwender aus diesen Grundoperationen aufgebaut werden und in Unterprogrammen (Routinen) niedergelegt werden.

Bedingungsbits (Flags, Zustandsbits)

Die ALU gibt über 5 "Bedingungsbits" (Flags) Auskunft über bestimmte Ergebnisse von arithmetischen und logischen Operationen. Diese 5 Flags signalisieren folgende Ergebnisse bei Operationen: Vorzeichen, Null, Parität, Übertrag und Hilfsübertrag.
Es ist möglich, Programmsprünge vom Zustand einer oder mehrerer Bedingungsbits abhängig zu machen. So kann das Programm beispielsweise die Anweisung enthalten, in ein spezielles Unterprogramm zu springen, wenn im Anschluß an eine Addition ein Übertrags-Flag gesetzt wird: "Bedingte Sprungbefehle".
Die 5 Bedingungsbits des Mikroprozessors 8080 sind:

S: Sign-Flag (Vorzeichenbit). Bei der Darstellung negativer Zahlen durch Bitmuster hatten wir gesehen, daß man bei Verwendung der Zweierkomplement-Darstellung das vorderste Bit eines Datenwortes als Vorzeichen-Indikator verwendet: hat es den Wert 1, so wird die Zahl als negativ angesehen (-1 bis -128), hat es den Wert 0, so wird die Zahl als positiv interpretiert (0 bis + 127). Das Vorzeichenbit gibt nach einer Operation den Wert dieses vordersten Bits wieder.

Z: Zero-Flag (Nullbit). Dieses Bedingungsbit wird dann gesetzt, wenn das Ergebnis einer arithmetischen oder logischen Operation

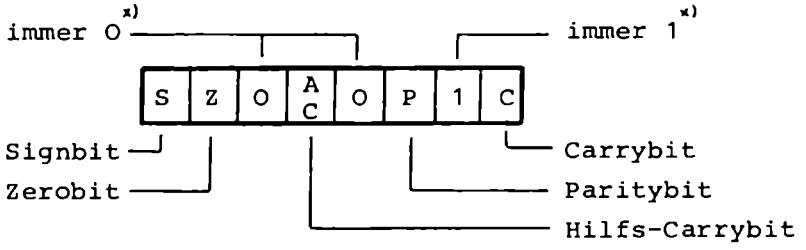
null ist. Das Nullbit wird rückgesetzt, wenn das Ergebnis dieser Operation ungleich null ist.

P: Parity-Flag (Paritätsbit). Nach arithmetischen und logischen Operationen wird eine Paritätsprüfung des Ergebnisses vorgenommen. Die Anzahl der Einsen im Ergebnis-Byte wird dabei gezählt: ist die Anzahl gerade, wird das Paritätsbit gesetzt, ist die Anzahl ungerade, wird das Paritätsbit rückgesetzt.

C: Carry-Flag (Übertragsbit). Eine Operation mit Überlauf des Ergebnisses an der höchsten Stelle setzt das Übertragsbit, eine Operation ohne Überlauf setzt das Übertragsbit zurück. Die Operationen, die das Carry-Flag verändern können, sind: Addition, Subtraktion, zyklisches Schieben und logische Operationen.

AC: Auxiliary-Carry-Flag (Hilfs-Übertragsbit). Das Hilfs-Carry-Flag zeigt den Überlauf aus dem ersten Halb-Byte an (von der 4. in die 5. Stelle). Wird nur beim Befehl DAA verwendet.

Die 5 Flags sind in einem Flag-Register F (5 Flag-Flipflops) zusammengefaßt, dessen Inhalt in einem 8-Bit-Wort abgefragt werden kann. Das Format dieses Flag-Bytes ist:



Universelle Register

Die universellen Register stehen dem Anwender zur vorübergehenden Speicherung von Daten oder Zwischenergebnissen innerhalb des Zentralprozessors zur Verfügung.
Der Mikroprozessor 8080 besitzt 7 universelle Register mit 8 Bit Breite, die auch paarweise angesprochen werden können und dann 16-Bit-Daten (z. B. Adressen) aufnehmen können. Diese 7 Re-

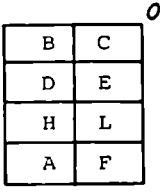
*) beim Mikroprozessor 8085 sind diese 3 Füll-Bits "undefiniert"

gister werden über die Buchstaben A,B,C,D,E,H,L angesprochen.

A steht dabei für "Akkumulator". Der Akkumulator hat als Register eine Sonderstellung, da er unmittelbar mit der ALU in Verbindung steht, während die übrigen Register nur über Multiplex-Betrieb erreichbar sind (siehe Blockschaltbild Abb. 5). Der Akkumulator speichert normalerweise einen der Operanden, die von der Arithmetik-Logik-Einheit verarbeitet werden. Ein Befehl kann z.B. die ALU veranlassen, den Inhalt eines anderen Registers zum Inhalt des Akkumulators zu addieren und das Ergebnis im Akkumulator zu speichern. Allgemein ist der Akkumulator sowohl die Quelle für einen Operanden als auch das Register, in dem das Ergebnis einer arithmetisch-logischen Operation gespeichert wird.

Wenn die Register paarweise angesprochen werden sollen, also als 16-Bit-Registerpaare, so besteht folgende Zuordnung:

Registerpaar B:	besteht aus	Register B und	Register C
Registerpaar D:	besteht aus	Register D und	Register E
Registerpaar H:	besteht aus	Register H und	Register L
Registerpaar PSW:	besteht aus	Register A und	Flagregister F



Dabei bezieht sich das Registerpaar PSW ("Processor Status Word" oder "Program Status Word") auf den Akkumulator und das Flagbyte, PSW beschreibt also wesentlich den momentanen Zustand des Mikroprozessors.

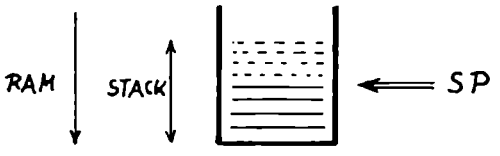
Das Registerpaar H (H und L) hat eine hervorgehobene Stellung bei den Programm-Befehlen. Wird eine Speicheradresse in einem Befehl indirekt durch ein Registerpaar angegeben, so muß diese Speicheradresse in H und L stehen (H enthält dabei das höherwertige Byte, L das niederwertige Byte der Adresse). Als Beispiel dient der uns schon bekannte Befehl INR M: Die Adresse des zu inkrementierenden Speicherplatzes M findet der Mikroprozessor in den Registern H,L ("indirekte Adressierung").

Spezielle Register

Befehlsregister (Instruction Register), 8 Bit: Der Operationscode eines vom Speicher abgerufenen Befehls wird im Befehlsregister des Mikroprozessors gespeichert, bis er dekodiert und ausgeführt ist. Handelt es sich um einen Mehrbyte-Befehl, so werden die Operanden in zusätzlichen 8-Bit-Zwischenregistern W und Z abgelegt.

Befehlszähler oder Programmzähler (Program Counter, PC), 16 Bit: In diesem 16-Bit-Register steht die Adresse des nächsten auszuführenden Befehls. Im Normalfall wird der Befehlszähler nach jedem eingeholten Befehlsbyte automatisch um 1 erhöht.

Stapelzeiger (Stack Pointer, SP), 16 Bit: Der Stackpointer zeigt den zuletzt besetzten Platz im Stack an, d.h. er ist ein 16-Bit-Register, das diejenige RAM-Adresse enthält, auf der zuletzt in den Stapelspeicher eingeschrieben wurde.



Bedingungs-Register (Flag-Register, F), 5 Bit: Es enthält den Wert der 5 Bedingungsbits S, Z, AC, P, C und kann über das beschriebene Flagbyte F abgefragt werden.

Steuerwerk

Die Steuerlogik ist eine besonders wichtige Funktionseinheit des Mikroprozessors. Sie sorgt für die ordnungsgemäße Folge von Abläufen bei der Durchführung eines Befehls, d.h. für die Umwandlung des 1. Befehls-Bytes (Operationscode) in einen Operationsablauf innerhalb des Mikroprozessors und gegebenenfalls auch außerhalb (z.B. veranlaßt das Steuerwerk die notwendigen Einzelschritte bei der Durchführung des Befehls MOV B,A).

Der auszuführende Befehl wird vom Steuerwerk aus dem Speicher in das Befehlsregister (Operationscode) bzw. in die zusätzlichen Zwischenregister W,Z (Operanden) eingelesen. Im Befehlsdekoder wird das Operationscode-Bitmuster dekodiert, d.h. der "Inhalt" des Befehls in Steuersignale umgewandelt, die die zeitliche und lo-

gische Steuerung der Befehlsausführung bewirken: Befehlszyklus-steuerung.

Der Mechanismus, mit dem der Prozessor ein Befehlswort in bestimmte Abläufe umwandelt, ist kompliziert und wird hier nur angedeutet: Die 8 Bits, die jeweils in dem Befehlsregister gespeichert sind, werden dekodiert und aktivieren selektiv eine von mehreren Leitungen innerhalb des Mikroprozessors. Im vorliegenden Fall (8 Bit) können es bis zu 256 Leitungen sein. Jede Leitung dient einer Reihe von Abläufen zur Durchführung eines bestimmten Befehls. Eine aktivierte Leitung kann beispielsweise mit Zeitimpulsen beaufschlagt werden, mit denen elektrische Signale für spezielle Abläufe erzeugt werden.

Es läuft also innerhalb des Mikroprozessors ein "Mikroprogramm" ab zur Ausführung des Befehlsinhaltes. Dieses interne Mikroprogramm für jeden Befehl dauert je nach Befehl 4 - 18 Taktperioden und ist natürlich vom Hersteller des Mikroprozessors fest eingegeben und vom Anwender nicht beeinflußbar. (Es gibt jedoch spezielle Mikroprozessor-Typen, bei denen man sich den Befehlssatz und das zugehörige interne Mikroprogramm selbst herstellen kann: Man nennt dann solche Mikroprozessoren "mikroprogrammierbar", ein äußerst raffinierter und komplizierter Vorgang, der bei Bit-Slice-Mikroprozessoren möglich ist.)
Der Mikroprozessor 8080 ist nicht mikroprogrammierbar.

5.13) Systemsteuerungseinheit 8228

Die 8 Datenleitungen des Mikroprozessors 8080, sowie die meisten Steuerleitungen werden der Systemsteuerungs-Einheit zugeführt und dort in die endgültige Form des Datenbus und "Steuerbus" gebracht. Dabei wird beim Mikroprozessor 8080 auch das Zustands-Wort (Status-Wort), das zu Beginn jedes Maschinenzyklus kurzzeitig auf dem 8o8o-Datenbus ausgesandt wird, an die Systemsteuerungseinheit übermittelt und dort für den Steuerbus weiter verarbeitet.

Die Systemsteuerungseinheit 8228 besteht aus dem Zweiweg-Datenbus-Treiber und der Systemsteuerungslogik.

Der Datenbus-Treiber dient dazu, die über den bidirektionalen Datenbus laufenden Daten zwischen Mikroprozessor 8080 und Speicher- bzw. E/A-Bausteinen durchzuschalten (zu "puffern"). Die Treiberschaltungen sind Tri-State-Puffer, d.h. außer in die beiden Zustände High und Low können diese Verstärker durch Steuersignale (z.B. HLDA) auch in einen hochohmigen Zustand versetzt werden, was bedeutet, daß der Datenbus vom Mikroprozessor elektrisch abgetrennt wird. Dadurch ist "direkter Speicherzugriff" (DMA) durch ein peripheres Gerät möglich (siehe Kapitel 6).

Systemsteuerungslogik: Wie bereits ausgeführt wurde, gibt der Mikroprozessor 8080 zu Beginn eines jeden Maschinenzyklus kurzzeitig ein Status-Wort auf seinen Datenbus, das anzeigt, welche Operation während des betreffenden Maschinenzyklus stattfinden wird. Hieraus werden durch logische Verknüpfung mit 2 weiteren Signalen des Mikroprozessors 8080 (DBIN und $\overline{WR}$ *) 4 Steuersignale gebildet. Diese 4 Steuersignale werden auf 4 Leitungen ausgesandt, die den wichtigsten Teil des Steuerbus des Systems 8080 ausmachen und der gezielten Aktivierung der Speicher-Bausteine bzw. der Ein/Ausgabe-Bausteine dienen.

Die 4 Steuersignale lauten:

- $\overline{MEMR}$: Steuerbefehl zum Lesen des Speichers (RAM oder ROM): Memory Read
- $\overline{MEMW}$: Steuerbefehl zum Einschreiben in den Speicher (RAM): Memory Write
- $\overline{I/OR}$: Steuerbefehl zum Lesen der Eingangsdaten der Ein/Ausgabe-Bausteine: In/Out Read
- $\overline{I/OW}$: Steuerbefehl zum 'Schreiben' der Ausgangsdaten der Ein/Ausgabe-Bausteine: In/Out Write

Der Inversionsstrich über den Abkürzungen bedeutet, daß diese Steuersignale "Low-aktiv" sind, d.h. daß der 0-Zustand des Steuersignales die aktive Phase des Signales darstellt.

---

*) DBIN = Data Bus In (Daten lesen)
WR = Write (Daten schreiben)

5.14) Zusammenstellung der Leitungen vom und zum Zentralprozessor

Beim System 8080 steht die CPU über folgende Leitungen mit dem übrigen System in Verbindung:

Adressbus: 16 Adressleitungen A_0- A_{15}, die unmittelbar aus dem Mikroprozessor 8080 kommen.

Datenbus: 8 bidirektionale Datenleitungen D_0- D_7, die vom Bus-Treiber der Systemsteuerungseinheit kommen.

Eine Reihe von Steuerleitungen, die gezielt an die einzelnen Systembausteine angeschlossen werden. Es sind dies:

- der eigentliche "Steuerbus" mit den Leitungen: $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, $\overline{\text{I/OR}}$, $\overline{\text{I/OW}}$.
- Signalleitungen, die eine Programmunterbrechung (Interrupt) ermöglichen: INT (Interrupt-Anforderung), INTE (Interrupt Enable, Interrupt-Bereitschaft), $\overline{\text{INTA}}$ (Interrupt Acknowledge, Interrupt-Bestätigung).
- Signalleitungen, die einen Wartezustand des Prozessors ermöglichen: RDYIN (Ready In, Warte-Anforderung), WAIT (Wartezustand-Anzeige).
- Ein Signal-Eingang, der einen direkten Speicherzugriff (Direct Memory Access: DMA) ermöglicht: HOLD (Anhalten).
- Der System-Rücksetz-Signaleingang: $\overline{\text{RESIN}}$ (Reset In).
- Ein asynchroner Bus-Freigabe-Eingang, dessen Signal die Datenbus-Ausgänge und Steuersignal-Ausgänge in den hochohmigen Zustand versetzt und so einen direkten Zugriff eines Peripherie-Elementes auf den Datenbus ermöglicht: $\overline{\text{BUSEN}}$ (Bus Enable, Bus-Freigabe).
- Ein Ausgang für die Oszillatorfrequenz: OSC.
- Ein Ausgang für das Ø2-Taktsignal mit TTL-Pegel.

Die meisten dieser speziellen Steuersignale werden im Kapitel 6 besprochen.

Spannungs-Versorgung: Der Mikroprozessor 8080 benötigt noch 3 Versorgungsspannungen: +12 V, +5 V, -5V.

5.2) Speicher

Der Gesamtspeicher setzt sich zusammen aus dem Festwertspeicher ROM und dem Schreib-Lese-Speicher RAM. Die vom Hersteller angebotenen Speicherbausteine enthalten nur eine begrenzte Anzahl von Speicherplätzen je Chip. Je nach benötigtem Gesamtspeicherbedarf setzt der Anwender seinen ROM- bzw. RAM-Speicher aus mehreren Einzelbausteinen zum Gesamtspeicher zusammen. Es muß dann eine "Baustein-Auswahl-Logik" (Chip-Select-Logik) vorgeschaltet werden, die dafür sorgt, daß bei der Adressierung eines Speicherplatzes nur derjenige Chip angesprochen wird, auf dem sich der gewünschte Speicherplatz befindet.

Abb. 8 zeigt die Anordnung eines Gesamtspeichers, der aus 4 ROM- und 4 RAM-Bausteinen zusammengesetzt ist. Diese Speicheranordnung wird im folgenden beschrieben.

5.2 1) ROM

Der dargestellte ROM-Speicher besteht aus 4 Bausteinen, wobei entweder die maskenprogrammierten ROM-Bausteine 2316 E oder die elektrisch programmierbaren und UV-löschbaren EPROM-Bausteine 2716 eingesetzt werden können. Beide Versionen sind zueinander anschlußkompatibel ("pinkompatibel"), können also gegeneinander ausgetauscht werden.+) Für Anwender, welche ihre Programme selbst entwickeln wollen, sind die EPROM-Versionen besonders vorteilhaft, da hierbei anfängliche Programmierfehler wieder korrigiert werden können. Bei größeren Stückzahlen in Serienfertigungen werden dagegen meist maskenprogrammierte ROM-Bausteine verwendet, deren Programminhalt nicht mehr verändert werden kann.

Ein ROM-Baustein 2316 bzw. 2716 enthält 16 K Bit = 16 384 Bit Speicherelemente. Er ist 2048 x 8 Bit (2 K x 8) organisiert, ermöglicht also die Speicherung von 2048 Worten mit 1 Byte Länge. 8-Bit-Worte sind die kleinste adressierbare Einheit in diesem Speicherbaustein. Abb. 6 zeigt den Schaltungs-Aufbau eines EPROM-Speicherbausteins 2716.

+) Daneben stehen auch pinkompatible, elektrisch löschbare EEPROM-Bausteine zur Verfügung (z.B.die Typen 2816, 2817).

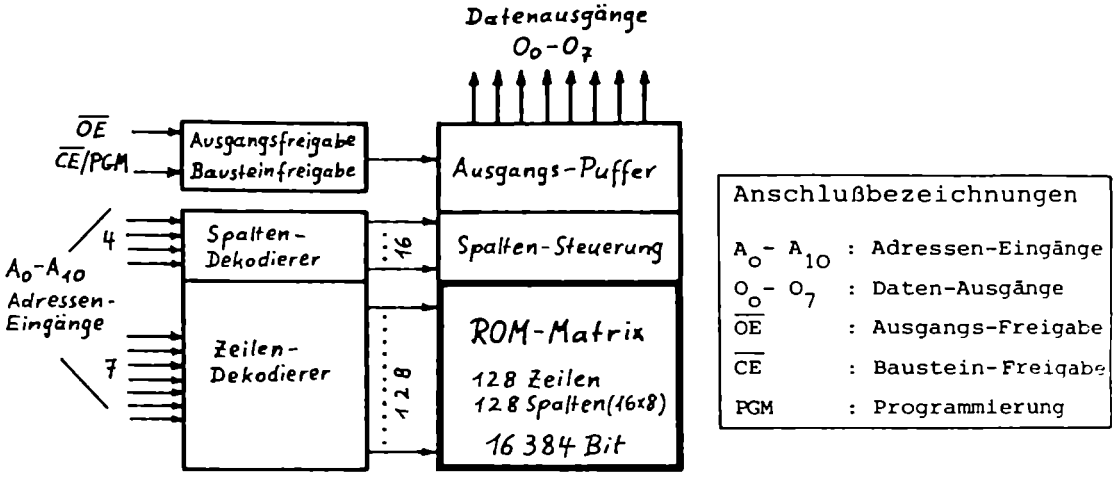


Abb. 6: Blockschaltbild des EPROM-Bausteins 2716
(ROM-Baustein 2316 E entsprechend)

Zur Adressierung der 2048 Speicherplätze eines Chips werden 11 Adressleitungen A_0- A_{10} benötigt (2^{11} = 2048). Diese Adressleitungen werden im Dekodierer des ROM-Bausteins so verknüpft, daß entsprechend der Zeilen-Spalten-Architektur des Chips jeder Speicherplatz eindeutig erreicht werden kann.

Mit 11 Adressleitungen ist jedoch nur der Speicherinhalt eines einzigen ROM-Bausteins 2316/2716 erfaßbar. Will man einen umfangreicheren Gesamtspeicher mit mehreren Bausteinen adressieren, so benötigt man weitere Adressleitungen A_{11}, A_{12} usw. Diese höherwertigen Bits der Adresse werden zur Auswahl der einzelnen Bausteine benützt. Für einen Gesamtspeicher aus insgesamt 4 ROM- und 2 x 2 RAM-Bausteinen, wie er in der Anordnung von Abb. 8 angenommen wurde, benötigt man also weitere 3 Adressleitungen A_{11}- A_{13}. Diese 3 zusätzlichen Adressleitungen werden zur Bildung von 2^3 = 8 verschiedenen Baustein-Auswahl-Signalen benützt. Die Speicherbausteine besitzen Baustein-Freigabe-Eingänge (CE bzw. CS), über die dann der jeweils gewünschte Baustein adressiert wird: ein Signal auf diesem Eingang aktiviert diesen speziellen Baustein. Die Bildung der Baustein-Auswahl-Signale aus den 3 Adressleitungen A_{11}- A_{13} erfolgt durch eine einfache logische Schaltung. Diese "Chip-Select-Logik" bewirkt, daß nur bei einer bestimmten

(von 8 möglichen) Adressen-Kombination ein aktivierendes Signal an einen bestimmten Baustein gelangt. Solche "1 aus 8 Dekoder" stehen dem Anwender auch kommerziell zur Verfügung in dem System-Baustein 8205, der in Abb. 8 eingezeichnet ist (siehe auch S.142). Mit den 4 ROM-Bausteinen 2316/2716 ist es also möglich, insgesamt 8K Byte-(8192 Byte-)Worte zu speichern. Die Adressen dieser Speicherplätze laufen hexadezimal von 0 bis 1FFF (0 bis 8191 dezimal).

Der Eingang $\overline{OE}$ (Ausgangs-Freigabe, Output Enable) des Bausteins 2716 gibt im 0-Zustand die Datenausgänge für das Lesen des ROM-Bausteins frei (Lesefreigabe).

Anschlußplan in der Gesamt-Speicheranordnung von Abb. 8: Für die 4 ROM-Bausteine ergibt sich im Gesamtspeicher folgender Anschlußplan:

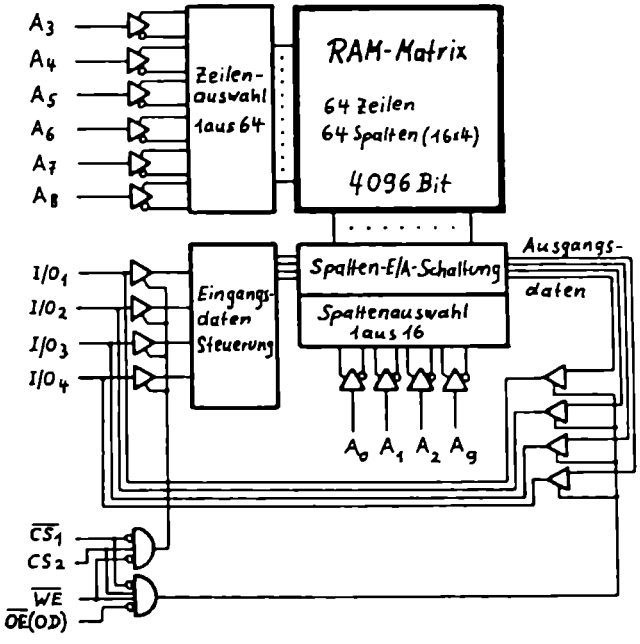
- Die 8 Datenleitungen D_0- D_7 des Datenbusses werden mit den Anschlüssen O_0- O_7 der ROM-Bausteine verbunden.
- Die 11 Adressleitungen A_0- A_{10} werden an die Anschlüsse A_0- A_{10} der ROM-Bausteine geführt.
- Aus den 3 Adressleitungen A_{11}- A_{13} werden durch eine Chip-Select-Logikschaltung (Baustein 8205) die Baustein-Auswahl-Signale gebildet, welche jeweils an den Baustein-Freigabe-Eingang $\overline{CE}$ (Chip Enable) der ROM-Bausteine geführt werden (Low-aktiv).
- Die Steuerleitung $\overline{MEMR}$, welche Low-aktiv den Lesevorgang des ROM einleitet, wird an den Anschluß $\overline{OE}$ (Output Enable) der ROM-Bausteine gelegt (Lesefreigabe).

Befehlsablauf: Der Zentralprozessor gibt zu Beginn eines Befehlsabrufs (Takt 1) eine Adresse aus. Der entsprechende Wert auf dem Adressbus führt zur Auswahl eines speziellen Speicherplatzes in einem bestimmten ROM-Baustein. Nach der Zugriffszeit des ROM erscheinen die Daten des adressierten Speicherplatzes an den Datenausgabe-Leitungen des Speichers. Zu diesem Zeitpunkt (Takt 3) befindet sich der Mikroprozessor-Datenbus im Eingabezustand, und der Prozessor übermittelt dem ROM-Baustein ein Speicher-Lesesignal ($\overline{MEMR}$) an den Lesefreigabe-Eingang $\overline{OE}$, das die adressierten Daten auf den Datenbus überträgt.

5.22) RAM

Für den Aufbau eines Schreib-Lese-Speichers RAM aus mehreren Bausteinen gilt im Prinzip das für ROM-Bausteine Gesagte mit dem Zusatz, daß eine weitere Steuerleitung $\overline{\text{MEMW}}$ für den Einschreibevorgang an den hierfür vorgesehenen RAM-Anschluß gelegt werden muß. Eine kleine Variante kommt dadurch herein, daß die vom Hersteller gelieferten RAM-Bausteine häufig 1 Bit oder 4 Bit organisiert sind, also mehrere Bausteine (8 bzw. 2) parallel geschaltet werden müssen, um 8-Bit-Datenworte abzuspeichern.*)

Der im Blockschaltbild Abb. 8 dargestellte RAM-Speicher besteht aus 4 statischen RAM-Bausteinen 2142. Ein Baustein 2142 enthält 4 K Bit = 4096 Bit Speicherelemente. Er ist 1K x 4 Bit organisiert, 2 parallel geschaltete Bausteine 2142 können also 1K = 1024 Datenworte mit 1 Byte Länge speichern. Zur Adressierung eines solchen "Baustein-Paares" werden 10 Adressleitungen benötigt (2^{10}= 1024). Um mehrere gleichartige Bausteine zu einem größeren RAM-Speicher zusammenzufassen, werden wieder weitere Adressleitungen zur Baustein-Auswahl herangezogen und mit den vorgesehenen Baustein-Freigabe-Eingängen (CS) verbunden. Mit den 2 x 2 RAM-Bausteinen 2142 von Abb. 8 ist es also möglich, insgesamt 2048 Datenworte mit 1 Byte Länge zu speichern.



Anschlußbezeichnungen	
A_0 - A_9	: Adressen-Eingänge
I/O_1-I/O_4	: Daten Ein-/Ausgänge
$\overline{CS\ 1}$,CS 2	: Bausteinauswahl
$\overline{WE}$	: Schreib-Freigabe
$\overline{OE}$ (=OD)	: Ausgangs-Freigabe

Abb.7: Blockschaltbild des RAM-Bausteins 2142

Abb. 7 zeigt den Schaltungs-Aufbau eines RAM-Speicherbausteins 2142, der in einem 20-Pin-Gehäuse untergebracht ist. Der $\overline{WE}$-Eingang (Write Enable) des Bausteins 2142 gibt im 0-Zustand die Möglichkeit des Einschreibens in den RAM-Baustein (Schreibfreigabe), der $\overline{OE}$-Eingang (Output Enable) gibt im 0-Zustand die Datenausgänge für das Lesen des RAM-Bausteins frei (Lesefreigabe).

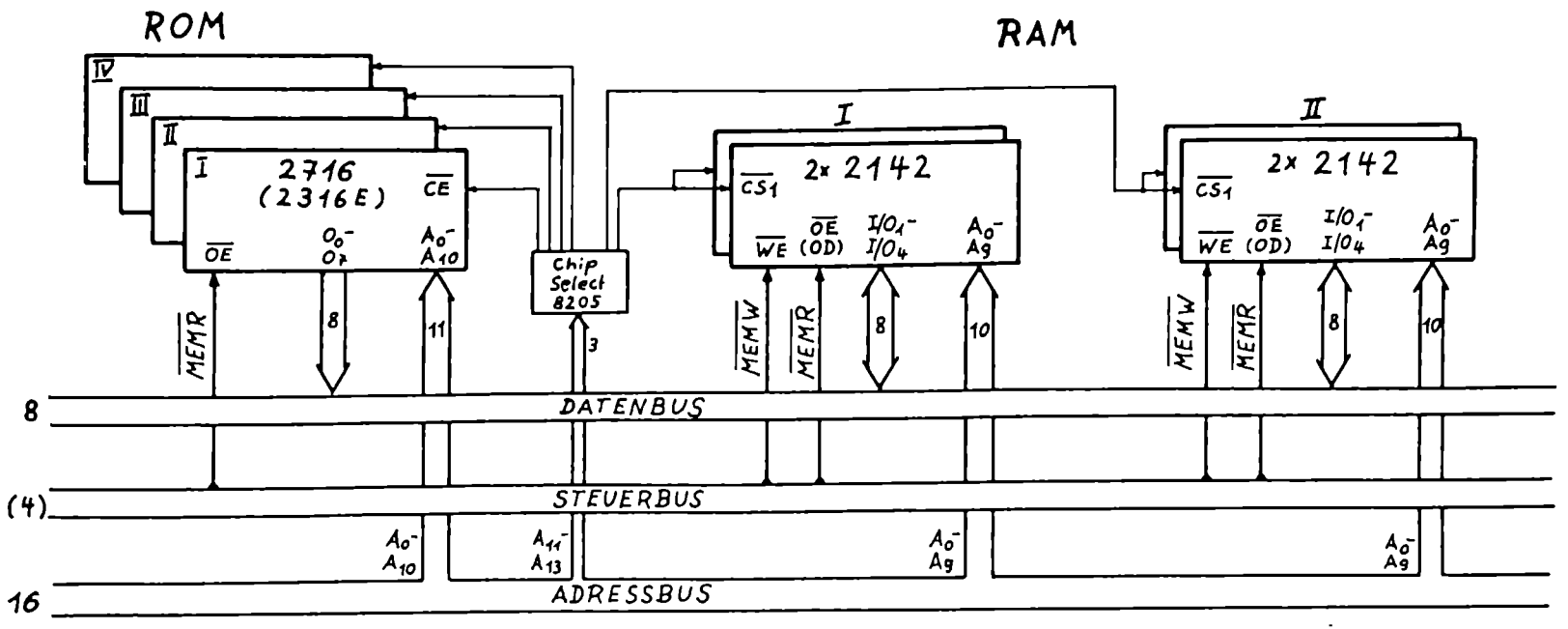
Anschlußplan für den RAM-Speicher in der Gesamt-Speicheranordnung von Abb. 8: Die RAM-Speicheradressen sollen an die ROM-Speicheradressen anschließen,d.h.hexadezimal von 2000 bis 27FF laufen; dies wird durch geeignete Wahl der Baustein-Auswahl-Signale erreicht. Für die 2x2 RAM-Bausteine 2142 ergibt sich folgender Anschlußplan:

- Die 8 Datenleitungen D_0- D_7 des Datenbusses führen an die 2 x 4 Anschlüsse I/O_1- I/O_4 der RAM-Baustein-Paare.
- Die 10 Adressleitungen A_0- A_9 des Adressbusses führen an die Anschlüsse A_0- A_9 der RAM-Bausteine.
- Die 3 Adressleitungen A_{11}- A_{13} werden wieder zur Erzeugung des Baustein-Auswahl-Signales verwendet, das an den Chip-Select-Eingang $\overline{CS1}$ der RAM-Bausteine gelegt wird.
- Die Steuerleitung $\overline{MEMR}$, welche den Lesevorgang des RAM einleitet, wird an den Eingang $\overline{OE}$ der RAM-Bausteine geführt.
- Die Steuerleitung $\overline{MEMW}$, welche den Schreibvorgang des RAM einleitet, wird an den $\overline{WE}$-Eingang der RAM-Bausteine geführt.

Befehlsablauf: Hinsichtlich eines Lesevorgangs behandelt der Mikroprozessor ein RAM in der gleichen Weise wie ein ROM. Ähnlich wie ein Lese- verläuft ein Schreibvorgang: Der Zentralprozessor gibt zu Beginn eines Speicher-Schreib-Zyklus (Takt 1) eine Adresse aus. Während Takt 3, wenn die einzuschreibenden Daten von dem Mikroprozessor ausgegeben werden und sich auf dem Bus stabilisiert haben, wird vom Prozessor ein Speicher-Schreib-Signal ($\overline{MEMW}$) erzeugt, das an den Schreibfreigabe-Eingang $\overline{WE}$ des RAM-Bausteins gelegt wird. Es blendet die Daten aus und leitet sie zum adressierten Speicherplatz im RAM.

---

*) Neuere Entwicklungen bei statischen RAM-Bausteinen tendieren zur 8-Bit-Organisation, wie z.B. die SRAMs 6116, 5517, 2128 usw., welche 2Kx8-Bit organisiert sind und gleiche Pin-Belegung wie der EPROM 2716 aufweisen, so daß ein bequemer Übergang von einem RAM-Baustein auf einen EPROM-Baustein möglich ist.



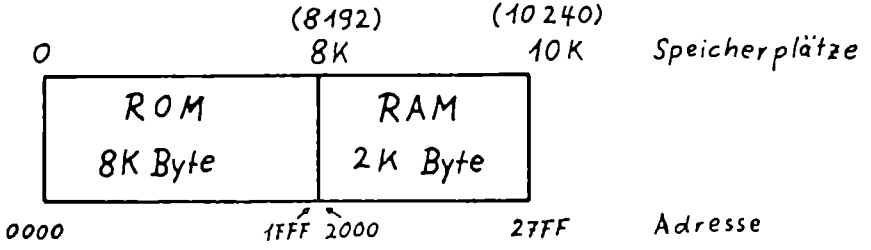


Abb.8: Blockschaltbild des Gesamtspeichers und Speicherplatz-Aufteilung

5.3) Eingabe/Ausgabe-Einheit (E/A bzw. I/O)

5.31) Parallele und serielle Datenübertragung

Für den Datenaustausch mit der Peripherie - z.B. einem externen Elektroniksystem oder mit Sichtgeräten und Druckern - besitzt das Mikrocomputersystem 8080 verschiedene E/A-Bausteine mit zwei grundsätzlich verschiedenen Möglichkeiten der Datenübertragung: 8-Bit-parallele Datenübertragung und serielle Datenübertragung.

Die parallele Datenübertragung erfolgt über E/A-Kanäle (I/O-ports) mit je 8 Leitungen, so daß jeweils 8-Bit-Daten parallel ausgetauscht werden können. Ein Beispiel für einen solchen E/A-Baustein mit paralleler Datenübertragung ist der Baustein 8255, der insgesamt 24 E/A-Anschlüsse enthält. Das Blockschaltbild von Abb. 9 zeigt den Aufbau dieses parallelen E/A-Bausteins ("PIO"). Die 24 Leitungen können in Gruppen zu 8 parallelen Leitungen und in zusätzliche Steuerleitungen zusammengefaßt werden. Die Übertragungsrichtung (Eingabe bzw. Ausgabe) innerhalb dieser Gruppen kann beim Baustein 8255 durch den Mikroprozessor programmiert werden.

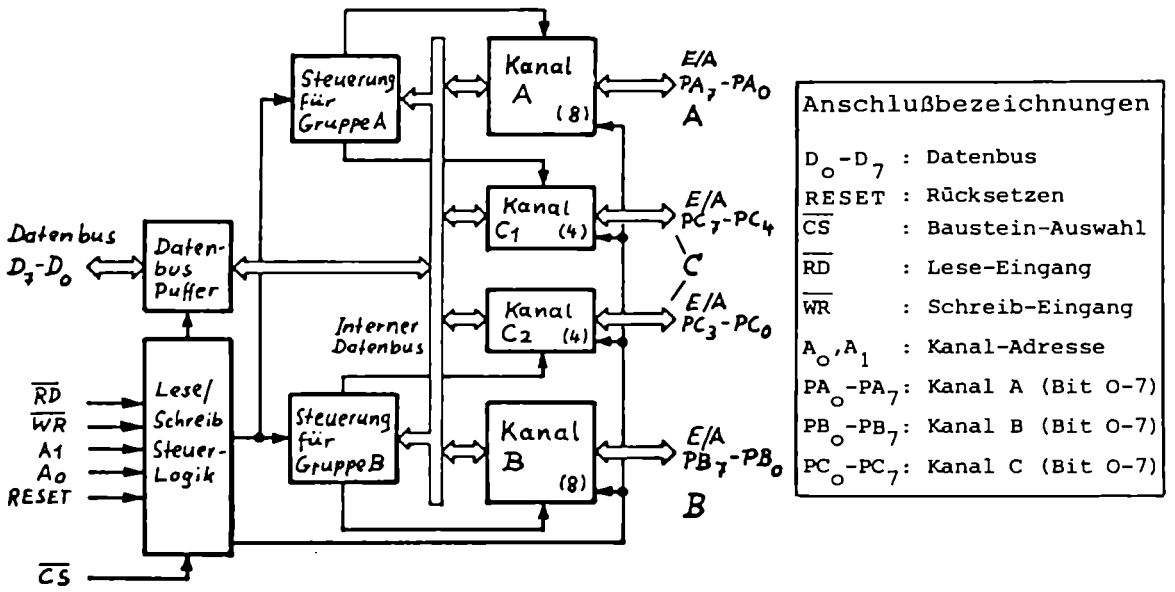
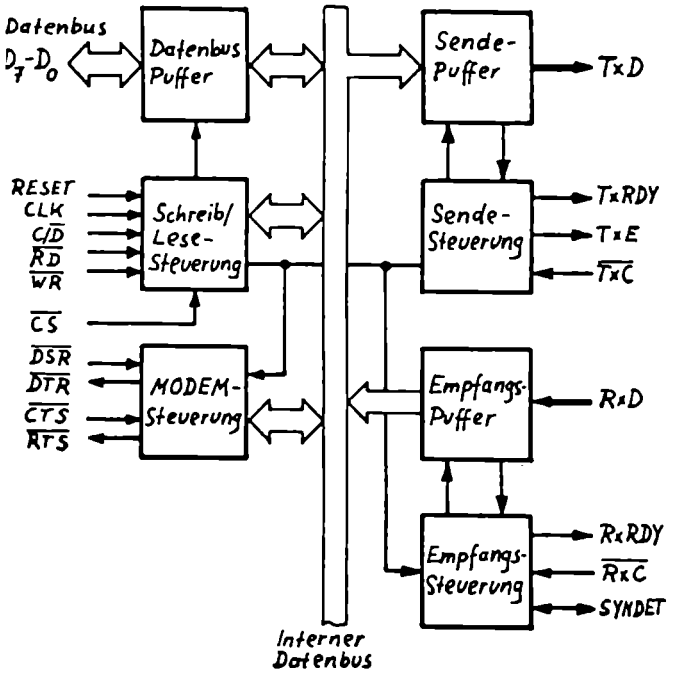


Abb.9: E/A-Baustein 8255 für parallelen Datenaustausch

Bei der seriellen Datenübertragung werden die aus dem Mikroprozessor kommenden 8-Bit-parallelen Daten über einen Zwischenspeicher in einen seriellen Datenstrom umgewandelt, der den E/A-Baustein auf einer einzelnen Leitung nach außen verläßt (synchron oder asynchron) - und umgekehrt werden serielle Daten, die von außen auf einer einzelnen Leitung an den E/A-Baustein gelangen, in parallele Daten für den Mikroprozessor umgewandelt. Ein Beispiel für einen seriellen E/A-Baustein ist der im Blockschaltbild Abb. 10 wiedergegebene Baustein 8251. Es ist ein "universeller Synchron/Asynchron-Sender/Empfänger-Baustein (USART)" für die Datenübertragung in Mikrocomputersystemen, der durch den Mikroprozessor programmiert werden kann und der für jeden heute zur Anwendung kommenden seriellen Datenübertragungsprozeß geeignet



Anschlußbezeichnungen

D_7-D_0	:	Datenbus
C/$\overline{D}$	:	Unterscheidung von Steuerinformation oder Daten
$\overline{RD}$	:	Befehl Daten lesen
$\overline{WR}$	:	Befehl Steuerinformation od.Daten schreiben
$\overline{CS}$	:	Baustein Freigabe
CLK	:	Takt Ø (TTL)
RESET	:	Rücksetzen
$\overline{TxC}$	:	Sende-Takt
TxD	:	Sende-Daten
$\overline{RxC}$	:	Empfangs-Takt
RxD	:	Empfangs-Daten
RxRDY	:	Empfänger bereit
TxRDY	:	Sender bereit
$\overline{DSR}$	:	Datenübertragung bereit
$\overline{DTR}$	:	Datenstation bereit
SYNDET	:	Synchronisationerkennung
$\overline{RTS}$	:	Sendeaufforderung
$\overline{CTS}$	:	Sendebereitschaft
TxE	:	Sendepuffer leer

Abb.10: E/A-Baustein 8251
für serielle Datenübertragung

ist. Der Sende-Puffer übernimmt Parallel-Zeichen vom Datenbus und wandelt sie in einen seriellen Bitstrom um, der Empfangs-Puffer nimmt serielle Daten auf, wandelt sie in parallele Daten um und schickt sie an den Mikroprozessor. Der interne Zeitablauf im seriellen E/A-Baustein 8251 wird durch das Taktsignal des Zentralprozessors gesteuert, die Daten-Übertragungsgeschwindigkeit vom und zum seriellen E/A-Baustein wird durch einen speziellen Generator für Sende- und Empfangstakt bestimmt (Baud-Rate-Generator).

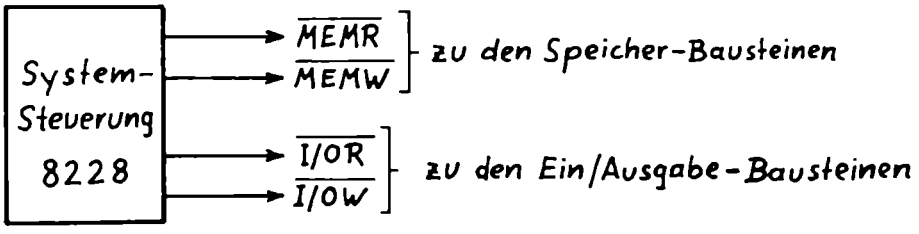
5.32) Adressierung von E/A-Kanälen

Adressierung mit "isolierter Eingabe/Ausgabe" (Standard-E/A)

Wie bereits erwähnt, erfolgt die Adressierung eines gewünschten E/A-Kanales - ähnlich der Adressierung eines Speicherplatzes - über den Adreßbus. Bei der üblichen "isolierten Eingabe/Ausgabe" (Isolated I/O), bei der nur die 8 niederwertigen Bits der Adresse (Kanalnummern) zur Kanalauswahl herangezogen werden, erfolgt die Unterscheidung der (möglichen) 256 Kanäle von den Speicherplätzen durch gesonderte Steuersignale für die E/A-Bausteine:

- das Steuersignal $\overline{I/OR}$ veranlaßt den "Lesevorgang" des E/A-Bausteins (d.h. Daten gelangen von der Peripherie zum Zentralprozessor): Eingabe
- das Steuersignal $\overline{I/OW}$ veranlaßt den "Schreibvorgang" des E/A-Bausteins (d.h. Daten gelangen vom Zentralprozessor nach außen zur Peripherie): Ausgabe

Diese Art der E/A-Adressierung bewirkt eine einfache Übertragung der Daten vom und zum Akkumulator (Register A). Denn unter Verwendung der Befehle IN oder OUT des 8080-Befehlssatzes verkehrt ein E/A-Baustein nur mit dem Akkumulator im Zentralprozessor.



Auswahl der E/A-Bausteine bei der Adressierung

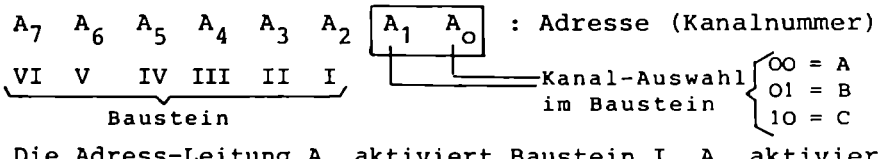
Da man bei einem einzelnen E/A-Baustein nur wenige Kanäle zur Verfügung hat, müssen bei einem größeren Bedarf an Kanälen mehrere Bausteine für die Gesamt-E/A-Einheit verwendet werden. Für die Adressierung eines bestimmten Kanals auf einem bestimmten Baustein muß also wieder — wie einst bei den Speicher-Bausteinen — eine Logik zur Baustein-Auswahl (Chip-Select-Logik) vorgesehen werden: Einige Bits der Adresse müssen zu Chip-Select-Signalen dekodiert werden.

Wenn man jedoch mit relativ wenig Kanälen (und damit mit wenig E/A-Bausteinen) auskommt, was bei vielen Anwendungen des Mikrocomputers der Fall ist, so kann man sich eines sehr einfachen Verfahrens zur Baustein-Auswahl bedienen, das im folgenden Beispiel erläutert wird:

Es werde eine Eingabe/Ausgabe-Einheit aus mehreren E/A-Bausteinen 8255 zusammengesetzt. Ein Baustein 8255 besitzt 3 Kanäle A,B,C, es genügen hierfür zur Adressierung 2 Bits der Adresse, z.B. A_1 und A_0. Die Zuordnung dieser Adressen zu den 3 Kanälen geschieht wie folgt:

OO bedeutet Kanal A
O1 bedeutet Kanal B
1O bedeutet Kanal C
(11 wird für das Steuerwort-Register des 8255 verwendet)

Die ganze Adresse (Kanalnummer) ist bei isolierter Eingabe/Ausgabe aber 8 Bit lang, es bleiben also für die Bausteine-Auswahl 6 Bit übrig. Jedem einzelnen dieser 6 Bit der Adresse wird nun direkt ein Baustein zugeordnet:



Die Adress-Leitung A_2 aktiviert Baustein I, A_3 aktiviert Baustein II usw.

Man kann also durch diese simple direkte Zuordnung von Adress-Leitung zu Baustein 6 verschiedene Bausteine eindeutig aufrufen,

ohne daß eine weitere Kodierung nötig wäre. Dies bedeutet in unserem Beispiel, daß 6 E/A-Bausteine 8255 à 3 Kanäle, d.h. insgesamt 18 Kanäle mit diesem Verfahren aufgerufen werden können. Diese Anzahl von Kanälen reicht in vielen praktischen Fällen aus.

Man nennt dieses simple Verfahren der Bausteinauswahl: "Lineare Auswahl" von E/A-Bausteinen.

Anschlußplan für eine Eingabe/Ausgabe-Einheit

Im Blockschaltbild Abb. 11 ist eine Eingabe/Ausgabe-Einheit dargestellt, welche aus 2 parallelen E/A-Bausteinen 8255 und 1 seriellen E/A-Baustein 8251 besteht. Die Ansteuerung erfolgt über "isolierte Eingabe/Ausgabe" und die Bausteine-Auswahl (Chip Select) erfolgt nach dem eben beschriebenen Verfahren der "linearen Auswahl".

Damit ergibt sich folgende Zuordnung der Datenbus-, Adreßbus- und Steuerbusleitungen zu den E/A-Bausteinen:

- Die 8 Datenbusleitungen liegen an den Eingängen D_0 - D_7 der E/A-Bausteine.
- Die Adreßleitungen A_0 und A_1 liegen an den Kanal-Adreß-Eingängen der Bausteine.
- Die Adreßleitungen A_2, A_3, A_4 dienen der linearen Baustein-Auswahl: Sie liegen Low-aktiv an je einem Chip-Select-Eingang $\overline{CS}$ (A_2=0 aktiviert z.B. den rechten Baustein 8255 in Abb.11).
- Das Steuersignal $\overline{I/OW}$ liegt am Schreibeingang $\overline{WR}$ der Bausteine.
- Das Steuersignal $\overline{I/OR}$ liegt am Leseeingang $\overline{RD}$ der Bausteine.
- Der serielle Baustein 8251 benötigt weitere Steuersignale für seine Funktion: Ein Taktsignal Ø aus dem Zentralprozessor sowie ein Sende- und ein Empfangstaktsignal aus einem besonderen Generator (Baud-Rate-Generator).

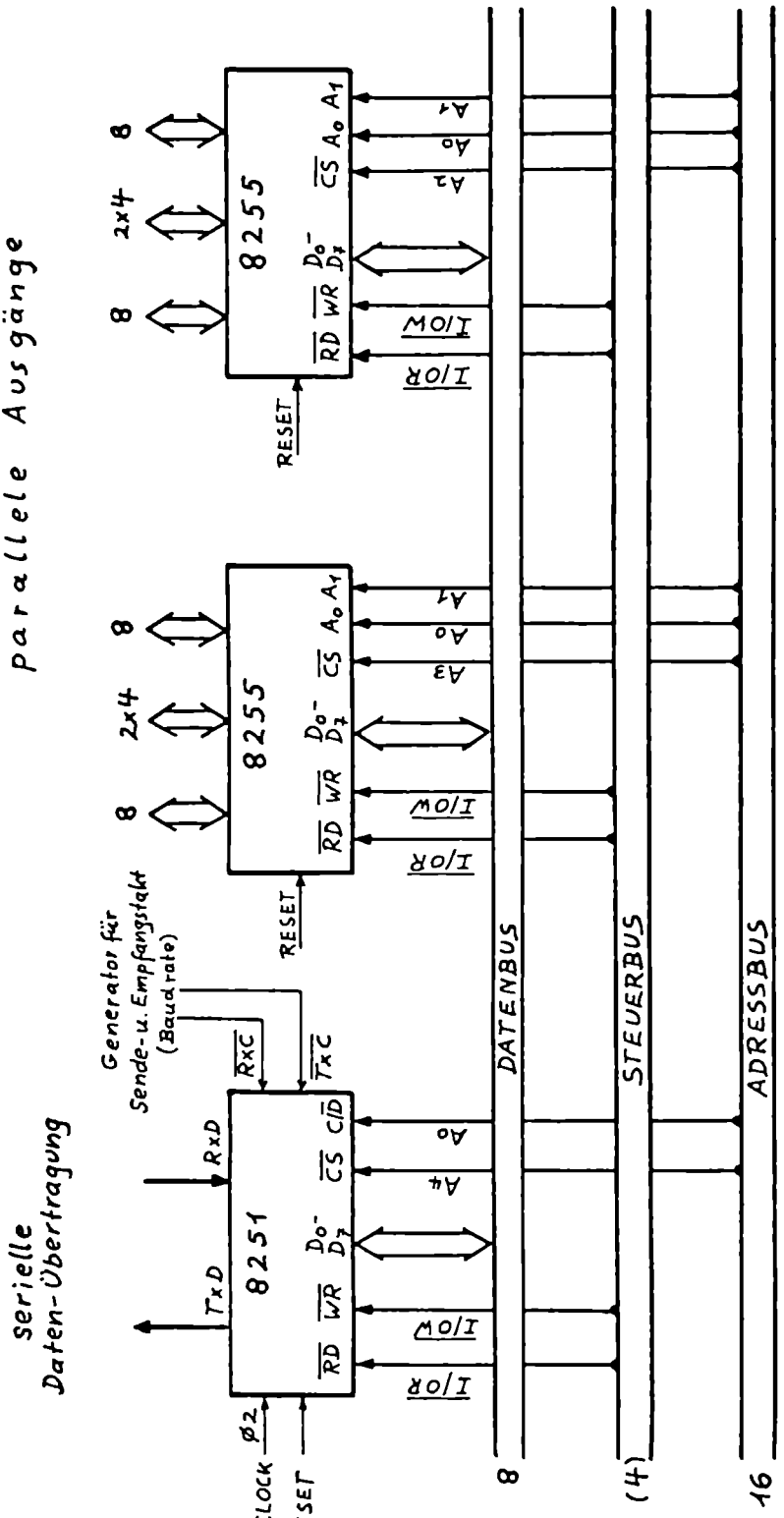


Abb.11: Eingabe/Ausgabe-Einheit mit 3 Bausteinen und linearer Baustein-Auswahl

Anhang: Adressierung der E/A-Kanäle mit
"Speicher-Eingabe/Ausgabe" (Memory-Mapped-I/O)

Wie bereits erwähnt besteht eine andere Methode der Adressierung einer Ein/Ausgabe-Einheit darin, die E/A-Kanäle wie Speicherplätze zu behandeln: Speicher-Eingabe/Ausgabe (Memory-Mapped-I/O). Mit dieser Methode ist eine softwaremäßig elegantere Durchführung von Eingabe/Ausgabe-Operationen möglich.

Durch die Reservierung eines Teiles der Speicher-Adressen für E/A-Kanäle lassen sich nämlich E/A-Bausteine mit den gleichen Befehlen bedienen, wie sie auch für die Speicher-Bausteine verwendet werden. Auf diese Weise ergibt sich ein "neuer" Befehlssatz für die Bedienung der E/A-Kanäle.

Wenn man das höchste Bit A_{15} der Adresse zur E/A-Kennzeichnung verwendet, erhält man eine einfache Zuordnung der E/A-Bausteine zur Adresse:

alle Adressen mit $A_{15} = 1$ bedeuten E/A-Kanäle,
alle Adressen mit $A_{15} = 0$ bedeuten Speicherplätze.

Die Adressleitung A_{15} muß also in diesem Falle so mit den Baustein-Freigabe-Eingängen logisch verbunden werden, daß $A_{15} = 1$ die Ein/Ausgabe-Bausteine aktiviert und $A_{15} = 0$ die Speicherbausteine. Als Steuersignale dienen nur noch $\overline{MEMR}$ und $\overline{MEMW}$.

Man kann jetzt praktisch alle Befehle des Mikroprozessors, die bisher nur zur Adressierung von Speicherplätzen zur Verfügung stehen, auch für E/A-Kanäle einsetzen (z.B. MOV, ADD, ANA etc.). Auch kann jetzt statt des Akkumulators als einzigem Übertragungsregister jedes beliebige interne Register des Mikroprozessors für die Übertragung von E/A-Daten verwendet werden. Man sieht, daß sich durch diese neue Art der Adressierung die Flexibilität des Mikrocomputer-Systems deutlich erhöhen kann.

Die Nachteile der Memory-Mapped-Eingabe/Ausgabe sind:
1.) Die Gesamtzahl der verfügbaren Speicherplätze wird reduziert.
2.) Memory-Mapped-E/A ist langsamer als isolierte E/A, da bei der Speicheradressierung 16-Bit-Adressen eingelesen werden

müssen, anstatt nur 8-Bit-Kanalnummern bei isolierter E/A. Z.B. benötigt der Mikrocomputer für den Transfer eines 8-Bit-Wortes zwischen Akkumulator und Ein/Ausgabe-Einheit
bei Memory-Mapped-E/A: 13 Takte (Befehl "LDA adr"),
bei isolierter E/A : nur 10 Takte (Befehl "IN nr").

Programmierung der Ein/Ausgabe-Bausteine

Die E/A-Bausteine 8251 (USART) und 8255 (PIO) sind in verschiedenen Betriebsarten verwendbar, die durch den Zentralprozessor programmiert werden können. Die detaillierte Beschreibung der vielfältigen Programmierungsmöglichkeiten der E/A-Bausteine 8251 und 8255 würde den Rahmen dieser Einführung übersteigen. Es wird hierzu auf die Literatur der Herstellerfirmen verwiesen (siehe Literaturverzeichnis am Schluß).

6) PROGRAMM-ABLÄUFE

6.1) "Normaler" sequentieller Programm-Ablauf

Bei den bisher betrachteten Programmbeispielen sind die einzelnen Befehle eines Programms in dem Speicher des Mikrocomputers in ihrer logischen Reihenfolge aufeinanderfolgend abgelegt. In dieser Reihenfolge werden die Befehle vom Zentralprozessor sequentiell abgerufen und synchron im Takt des Taktgenerators ausgeführt, bis der letzte Befehl eines Programms (die End-Adresse) abgearbeitet ist. Dies ist der einfache, "normale" Ablauf eines Programms.

Abweichungen von diesem einfachen Ablauf sind:

- Einfügen von Pausen in den synchronen Taktablauf: Wartezustand (Wait), Hold-Zustand.
- Abweichung vom sequentiellen Programmablauf: Programm-Sprünge (Jumps), Unterprogramme (Subroutines), Programm-Unterbrechungen (Interrupts).

Mit Hilfe dieser komplexeren Programm-Abläufe ist eine beträchtliche Steigerung der Effizienz des Mikrocomputers möglich. Die verschiedenen Möglichkeiten werden im folgenden näher erläutert.

6.2) Wartezustand (Speicher- bzw. E/A-Synchronisation)

Alle Operationen eines Mikrocomputers werden durch den zentralen Taktgenerator in ihrem Ablauf bestimmt. Die maximale Geschwindigkeit, mit der ein Zyklus ablaufen kann, wird mit bestimmt durch die Geschwindigkeit, mit der Daten mit dem Speicher bzw. der Eingabe/Ausgabe-Einheit ausgetauscht werden können (Zugriffszeit des Speichers bzw. der E/A-Einheit).

Wenn beispielsweise der Mikroprozessor eine Adresse an den Speicher geschickt hat, kann er zunächst nicht weiterarbeiten,

bis der Speicher reagiert hat. Die meisten Speicher sind zwar sehr viel schneller, als es der Ablaufzyklus erfordert; es gibt jedoch auch Speicher, die das adressierte Wort nicht innerhalb der Minimalzeit austauschen können, die sich aus der Taktperiode des Prozessors ergibt. Der Prozessor muß deshalb eine Synchronisationsanordnung enthalten, die es dem Speicher gestattet, einen sogenannten Wartezustand anzufordern. Wenn der langsame Speicher ein Lese- oder Schreib-Freigabesignal erhält ($\overline{\text{MEMR}}$ bzw. $\overline{\text{MEMW}}$), beaufschlagt er die Bereit- (RDYIN-) Leitung des Prozessors mit einem Low-Signal, das diesen zeitweilig stillstehen läßt. Erst wenn der Speicher genügend Zeit hatte zu reagieren, gibt er die RDYIN-Leitung des Prozessors wieder frei (H-Pegel), und der Zyklus läuft synchron weiter. Der Wartezustand des Prozessors kann am Anschluß "WAIT" geprüft werden.

Eine Warte-Anforderung wird oft auch von einem E/A-Element erteilt, das langsamer als der Mikroprozessor arbeitet. Die Steuerlogik läßt auf diese Anforderung hin den Mikroprozessor stillstehen, bis der E/A-Baustein seine Daten bereit hat.

Die Warte-Anforderung ist also kein Befehl, sondern ein hardwaremäßiges direktes Spannungssignal an den Warte-Anforderungs-Eingang RDYIN des Zentralprozessors!

6.3) Hold-Zustand (direkter Speicherzugriff: DMA)

Die Leistungsfähigkeit eines Mikrocomputers kann durch den Einsatz eines HOLD-Signales, d.h. das Anhalten eines Prozessors, erhöht werden. Dadurch wird "direkter Speicherzugriff" möglich (Direct Memory Access = DMA).

Normalerweise überwacht der Zentralprozessor selbst alle Datenübertragungen. Das heißt, Daten von peripheren Geräten, die in den Speicher eingegeben werden sollen, werden vom E/A-Baustein zunächst zum Zentralprozessor (Akkumulator-Register) übertragen und von dort zu der vorgesehenen Speicherstelle. Auf ähnliche

Weise gelangen Daten vom Speicher über den Prozessor zur E/A-Einheit und von dort nach außen zur Peripherie.

Einige Peripherieelemente können die Übertragung von Informationen zum oder vom Speicher selbst sehr viel schneller durchführen als der Prozessor. Wenn größere Datenmengen zu übertragen sind, läßt sich der Durchsatz des Systems dadurch steigern, daß man der Peripherie eine direkte Übertragung der Daten ermöglicht. Der Zentralprozessor muß hierzu seine Operationen vorübergehend unterbrechen, um ein doppeltes Ansprechen des Speichers durch Prozessor und Peripherie-Gerät zu vermeiden. Aus diesem Grund ist beim Mikroprozessor 8080 die Möglichkeit eines Anhalte-Zustandes (Hold-Zustand) vorgesehen, der durch ein H-Signal auf der HOLD-Leitung ausgelöst wird.

Auch dieses HOLD-Signal ist also kein Befehl, sondern ein hardwaremäßiges direktes Spannungssignal an den HOLD-Eingang des Mikroprozessors. Es soll schon an dieser Stelle darauf hingewiesen werden, daß dieses HOLD-Signal nicht mit dem später zu behandelnden (softwaremäßigen) HALT-Befehl "HLT" verwechselt werden darf.

6.4) Programm-Sprünge (Jumps)

Die Befehle, aus denen sich ein Programm zusammensetzt, sind im Programmspeicher des Mikrocomputers niedergelegt und mit definierten Adressen versehen. Der Zentralprozessor enthält in seinem 16-Bit-Befehlszähler-Register jeweils die Adresse des nächsten auszuführenden Befehls. Da die Befehle eines Programms in ihrer logischen Reihenfolge im Speicher hintereinander abgelegt sind, erhöht der Zentralprozessor im Normalfall nach jeder Befehls-Einholung den Befehlszähler automatisch um 1 auf die nächste Befehlsadresse.

Der Programmierer kann jedoch durch Sprung-Befehle dafür sorgen, daß der "normale" Programmablauf unterbrochen wird und das Programm an einer beliebigen Speicheradresse fortgesetzt wird. Ein solcher Sprungbefehl (Jump) enthält immer die Adresse des näch-

sten auszuführenden Befehls. Während der Durchführung des Sprungbefehls ersetzt der Zentralprozessor den Inhalt seines Befehlszählers durch die im Sprungbefehl angegebene Adresse. Auf diese Weise wird das Programm an der angegebenen Stelle fortgesetzt und zwar wieder sequentiell in fortlaufender Reihenfolge der Befehle bis zum Ende oder einem evtl. vorkommenden weiteren Sprungbefehl.

Man unterscheidet bei den Sprungbefehlen zwischen "unbedingten" und "bedingten" Befehlen:

- Der Befehl für unbedingte Sprünge lautet: "JMP adr" (Jump unconditional to address). Das Programm wird bei der vorgegebenen Befehlsadresse fortgesetzt - unbedingt.
- Bei den Befehlen für bedingte Sprünge wird das Programm bei der vorgegebenen Adresse nur dann fortgesetzt, wenn die Zustandsbits (Flags) S,Z,P,C bestimmte Werte haben ("Bedingungsbits"). Solche bedingten Sprungbefehle sind z.B.

  "JM adr" (Jump if Minus): Springe nur dann, wenn das Vorzeichen-Bit S = 1 ist.

  "JNC adr" (Jump if No Carry): Springe nur dann, wenn das Übertrags-Bit C = 0 ist.

  Die weiteren bedingten Sprungbefehle werden später im Befehlssatz des 8080 behandelt (Kapitel 7).

6.5) Unterprogramme (Subroutines)

Der Begriff Unterprogramm soll am Beispiel der häufig benötigten Multiplikations-Operation erläutert werden.

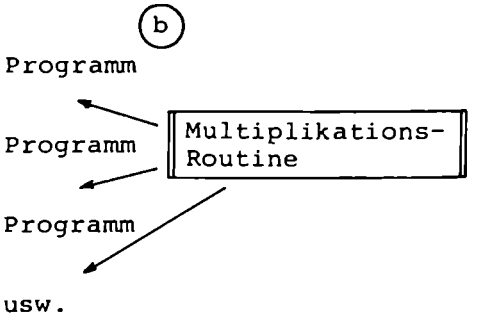
Der Mikroprozessor 8080 verfügt über keinen Befehl für eine Multiplikation, bietet aber die Möglichkeit, ein Byte zu einem anderen zu addieren. Man kann daher eine Multiplikation ausführen, indem man derartige Additionen mehrere Male hintereinander ausführt. Will man eine Multiplikation an mehreren Stellen des Programms durchführen, so müßte man an jeder dieser Stellen die

ganze Reihe der eben genannten Befehle (die "Multiplikations-Routine") einfügen: Beispiel a). Dazu wäre natürlich sehr viel Speicherplatz nötig.

(a)

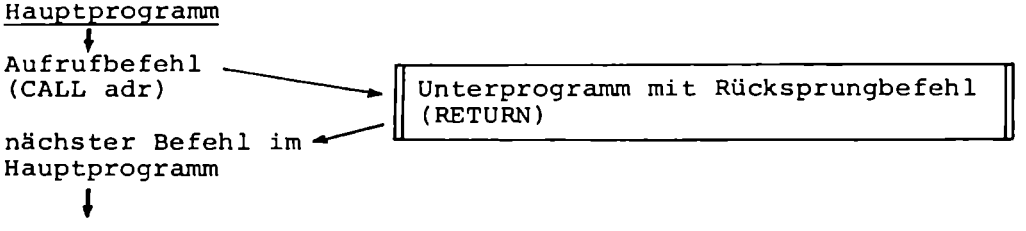
Programm
    Multiplikations-Routine
Programm
    Multiplikations-Routine
Programm
    Multiplikations-Routine
usw.

(b)



Da die Multiplikations-Routine immer gleich bleibt, ist es unnötig, diesen Programmteil immer wieder neu zu schreiben. Es ist besser, ihn nur ein einziges Mal abzuspeichern und jedes Mal aufzurufen und auszuführen, wenn er benötigt wird: Beispiel b). Eine solche immer wieder aufrufbare Routine nennt man ein Unterprogramm (Subroutine). Ein Unterprogramm ist also ein in sich abgeschlossenes kleines Programm, das sich an irgend einer Stelle im Speicher befinden kann.

Der Befehlssatz des Mikroprozessors 8080 besitzt Befehle zum Aufruf von Unterprogrammen und zum Rücksprung in das Hauptprogramm. Im einzelnen sieht der Programmablauf unter Verwendung von Unterprogrammen folgendermaßen aus:



Der Aufruf eines Unterprogramms erfolgt durch CALL-Befehle unter Angabe einer Adresse. Wenn das Programm ein Unterprogramm aufruft, muß der Mikroprozessor den Inhalt des Befehlszählers zur Zeit des Sprungs festhalten. Dadurch kann er das Hauptprogramm an der richtigen Stelle fortsetzen, sobald der letzte Befehl des

Unterprogramms ausgeführt ist. Damit eine ordnungsgemäße Rückkehr zum Hauptprogramm möglich ist, werden Unterprogramme vom Zentralprozessor wie folgt abgewickelt: Wenn der Prozessor einen CALL-Befehl ausführt, erhöht er die Zahl im Befehlszähler um 1 und speichert den Inhalt des Befehlszählers im Stapelspeicher (Stack). (Der Stapelspeicher enthält also jetzt die Adresse desjenigen Befehls, der nach Beendigung des Unterprogramms als nächster ausgeführt werden soll.) Anschließend lädt der Prozessor die Adresse, die in dem CALL-Befehl enthalten ist, in seinen Befehlszähler. Der nächste eingeholte Befehl ist infolgedessen der erste Schritt des Unterprogramms. Der letzte Befehl in jedem Unterprogramm ist ein Rückkehr-Befehl (RETURN), der keine Adresse zu enthalten braucht. Denn wenn der Prozessor einen Rückkehr-Befehl erhält, ersetzt er einfach den Inhalt seines Befehlszählers durch die im Stapelspeicher befindliche oberste Adresse. Hierdurch wird der Prozessor zur Wiederaufnahme des Hauptprogramms an der richtigen Stelle veranlaßt.
Unterprogramme werden oft ineinander verschachtelt, d.h. ein Unterprogramm kann einen Rufbefehl für ein zweites Unterprogramm geben usw. Die maximale Tiefe der Verschachtelung wird durch den für den Stapelspeicher reservierten RAM-Bereich bestimmt. Die Reihenfolge der Verschachtelung bleibt durch den schon erwähnten LIFO-Betrieb (Last In- First Out) des Stapelspeichers erhalten.

Sowohl der Unterprogramm-Aufruf CALL als auch der letzte Befehl eines Unterprogramms, der Rücksprungbefehl RETURN, können unbedingt oder bedingt sein:

- Unbedingte Befehle sind: "CALL adr" bzw. "RET".
- Bei den bedingten Befehlen wird der Aufruf- bzw. Rücksprungbefehl vom Zustand der Flags (S,Z,P,C) abhängig gemacht, z.B.:
  "CZ adr" (Call if Zero): Ins Unterprogramm gesprungen wird nur, wenn das Null-Bit Z = 1 ist.
- "RPE" (Return if Parity Even): Rücksprung erfolgt nur, wenn das Paritäts-Bit P = 1 ist.

Die weiteren bedingten CALL- und RETURN-Befehle werden später im Befehlssatz abgehandelt (Kapitel 7).

Ob sich ein Unterprogramm lohnt, hängt vom Speicherplatzbedarf ab. Bei längeren Routinen, die in einem Programm mehrmals vorkommen, lohnt sich vom Speicherplatz her die Erstellung eines Unterprogramms.

Es gibt allerdings nicht nur den Gesichtspunkt der Speicherplatz-Ökonomie, sondern auch den Gesichtspunkt der Ablaufgeschwindigkeit. Bei zeitkritischen Programmteilen (wenn also ein Computer gegenüber real ablaufenden Ereignissen nicht in Verzug geraten darf) wird man gelegentlich auf Unterprogramme verzichten, um die Aufruf- und Rückkehrbefehle einzusparen. Dann läuft das Programm schneller ab. Der Preis, den man dafür zahlt, besteht in größerem Aufwand an Speicherplätzen (siehe hierzu später auch die "Makro's").

6.6) Programm-Unterbrechungen (Interrupts)

Bei den meisten Mikroprozessoren ist die Möglichkeit für externe Programm-Unterbrechungen (Interrupts) gegeben, wodurch die Leistungsfähigkeit eines Mikrocomputers wesentlich erhöht werden kann. Die Unterbrechungsanforderung erfolgt z.B. von peripheren Geräten, die nur zu bestimmten Zeiten die Zusammenarbeit mit dem Mikrocomputer benötigen, oder von peripheren Geräten, die langsamer arbeiten, als der Zentralprozessor (z.B. Drucker, Magnetspeicher, Tastaturen etc). Der Grundgedanke einer solchen externen Programmunterbrechungsanforderung ist der, daß der Mikrocomputer nicht lange Zeit untätig in Warteposition verharren muß, sondern inzwischen an einem anderen Programm arbeiten kann und nur auf eine Unterbrechungsanforderung hin das periphere Gerät bedient.

Zum Beispiel kann der Mikrocomputer ein Datenwort innerhalb eines einzigen Maschinenzyklus an einen Drucker ausgeben, der Drucker braucht aber viele solcher Zyklen, um das entsprechende Zeichen auszudrucken. Inzwischen kann der Mikrocomputer sich anderweitig betätigen. Immer wenn der Drucker bereit ist, ein neues Datenwort zu empfangen, sendet er eine Unterbrechungsanforderung an den Zentralprozessor aus. Daraufhin unterbricht der Mikrocomputer die

Bearbeitung seines Hauptprogramms und wechselt in ein Unterprogramm über, das den Drucker bedient (Unterbrechungs-Service-Routine). Nach der Ausgabe des Datenwortes setzt der Mikrocomputer die Bearbeitung des Hauptprogrammes fort. Es ist zu beachten, daß dieser Unterbrechungs-Vorgang durch externe Hardware und nicht durch das Software-Programm veranlaßt wird.

Das System 8080 erlaubt eine hardwaremäßige Unterbrechung des laufenden Hauptprogramms (Background-Programm) durch hierfür vorgesehene Anschlüsse am Zentralprozessor. Die Unterbrechungsanforderung durch ein peripheres Gerät geschieht, indem das anfordernde Gerät die Unterbrechungs-Anforderungs-Leitung INT des Mikroprozessors auf H-Pegel bringt. Der Mikrocomputer 8080 entscheidet durch Programm-Befehle (softwaremäßig), ob er die Unterbrechungsanforderung akzeptieren will oder nicht ("maskierbarer" Interrupt). Dies geschieht durch die Befehle EI bzw. DI:

EI = Enable Interrupt (ermögliche Unterbrechung)
DI = Disable Interrupt (schließe Unterbrechung aus)

(Durch die Befehle EI oder DI wird ein Interrupt-Flipflop im Mikroprozessor gesetzt oder rückgesetzt; sein Zustand kann am Anschluß INTE des Mikroprozessors geprüft werden).

Man kann also in zeitlich kritischen Passagen des Programms vorübergehend die Unterbrechungsmöglichkeit ausschließen (durch DI) und danach wieder zulassen (durch EI).

Der Unterbrechungs-Anforderungs-Eingang INT ist asynchron. Eine Unterbrechungsanforderung darf daher jederzeit während eines beliebigen Befehlszyklus gestellt werden. Akzeptiert der Mikroprozessor eine Unterbrechungsanforderung von einem externen Gerät, so geschieht folgendes:

1.) der gerade vom Zentralprozessor in Bearbeitung befindliche Befehl wird zunächst vollständig zu Ende ausgeführt.

2.) Das Interrupt-Flipflop wird dann automatisch rückgesetzt (INTE = 0) und damit eine nochmalige Unterbrechung der Unterbrechung verhindert. Im folgenden Maschinenzyklus wird auf der Steuerleitung $\overline{\text{INTA}}$ vom Zentralprozessor ein Steuer-

signal (Low-Pegel) an das anfordernde Gerät ausgesandt und damit die Bereitschaft der CPU zur Unterbrechung bestätigt (Interrupt-Acknowledge).
Der Inhalt des Befehlszählers (das ist die nächste Befehlsadresse des Hauptprogramms) wird automatisch in den Stapelspeicher (Stack) gerettet.

3.) Das unterbrechende externe Gerät liefert über den Datenbus einen Befehl an die CPU, den diese ausführt. Dieser Befehl kommt also in diesem Fall nicht aus dem Speicher, sondern unmittelbar aus dem peripheren Gerät. Dieser Befehl ist beim Mikroprozessor 8080 einer von 8 möglichen RESTART-Befehlen (RST). Ein RESTART-Befehl ist ein 8-Bit-Befehl, welcher der CPU mitteilt, wo die Anfangsadresse für das gewünschte Unterbrechungs-Unterprogramm (Interrupt-Service-Routine) im Speicher liegt, welches z.B. die Ausgabe eines Datenwortes an einen Drucker steuert.

Die 8 RESTART-Befehle sind gezielt bestimmten Speicher-Adressen zugeordnet:

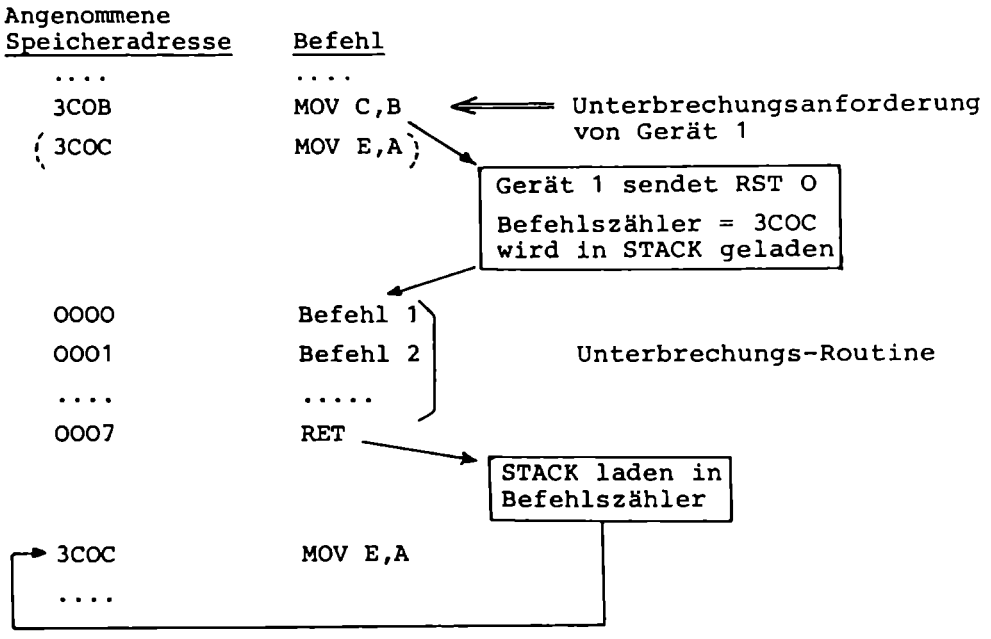
RST 0 bedeutet, daß das gewünschte Unterprogramm bei der Adresse 0 beginnt
RST 1 bedeutet die Anfangs-Adresse 8 (0008(H))
RST 2 bedeutet die Anfangs-Adresse 16 (0010(H))
RST 3 bedeutet die Anfangs-Adresse 24 (0018(H))
⋮
RST 7 bedeutet die Anfangs-Adresse 56 (0038(H))

Allgemein ausgedrückt bedeutet RST n die Anfangsadresse $8 \cdot n$ (wobei $0 \leq n \leq 7$ ist). An diesen reservierten Adressplätzen muß dann der 1. Befehl für die gewünschte Unterbrechungs-Service-Routine liegen. (Beliebt ist der RESTART-Befehl RST 7 mit dem einfachen Bitmuster 1111 1111, das leicht realisiert werden kann).

Die Interrupt-Service-Routinen sind also auf den ersten Blick nur 8 Byte lang (z.B. erfaßt RST 1 den Bereich von Adresse 8 bis Adresse 15 im Speicher). Jedoch kann jede Unterbrechungsroutine durch den Aufruf von Unterprogrammen beliebig erweitert werden.

4.) Der Mikrocomputer führt die vom peripheren Gerät durch RESTART-Befehl angeforderte Unterbrechungs-Routine durch. Nach erfolgter Ausführung der Unterbrechungs-Routine springt der Zentralprozessor in das Hauptprogramm zurück. Der letzte Befehl der Unterbrechungs-Service-Routine ist ein RETURN-Befehl, der die im Stack gespeicherte Befehlsadresse in den Befehlszähler zurückbringt, so daß das Hauptprogramm an der unterbrochenen Stelle fortgesetzt wird.
Spätestens vor dem Rückspringen ins Hauptprogramm muß durch den Befehl EI der Mikroprozessor wieder für neue Unterbrechungs-Anforderungen freigegeben werden.

Beispiel für einen Unterbrechungs-Ablauf:
Das externe Gerät 1 sendet eine Unterbrechungsanforderung an den Mikroprozessor 8080, während dieser gerade den Befehl mit der Adresse 3COB ausführt.



Dieser Befehl MOV C,B wird zu Ende ausgeführt und die Adresse 3COC wird in den Stack gebracht. Nachdem das externe Gerät 1 den Befehl RST O gesendet hat, wird der Programmzähler auf die erste Adresse 0000 der Unterbrechungsroutine gesetzt und die Routine durchgeführt.

Wenn der RETURN-Befehl ausgeführt wird, wird die Adresse 3COC vom STACK zurück in den Befehlszähler gebracht und das Programm an der Stelle fortgesetzt, bei der die Unterbrechung erfolgt ist.

Richtlinien für das Erstellen von Unterbrechungs-Service-Routinen:

Jedes Register und jedes Zustandsbit, das durch eine Unterbrechungsroutine verändert wird, muß zuvor sichergestellt werden und vor der Rückkehr ins unterbrochene Hauptprogramm wieder rückgeladen werden. Die Unterbrechungsroutine muß also als erstes den Inhalt des Akkumulators und der Flags (des Registerpaares PSW) sowie evtl. der übrigen Register in den STACK bringen durch PUSH-Befehle (z.B. "PUSH PSW" = Bringe den Inhalt des Registerpaares PSW in den STACK).

Dann kann das eigentliche Unterbrechungs-Unterprogramm ablaufen.

Zum Schluß werden vor dem RETURN-Befehl die abgelagerten Registerinhalte durch POP-Befehle wieder dem STACK entnommen und der alte Zustand von Akkumulator, Flags und evtl. übrigen Registern wiederhergestellt (z.B. "POP PSW" = Bringe die im STACK abgelagerten Daten zurück in das Registerpaar PSW).

Ferner wird, sofern nicht schon früher geschehen, wieder EI angeordnet.

Eine typische Unterbrechungsroutine sieht also folgendermaßen aus:

```
Operationscode  Operand     Kommentar
    PUSH        PSW         ; ZUSTANDSBITS UND AKKUMULATOR
    PUSH        B           ; SOWIE UEBRIGE REGISTERPAARE
    PUSH        D           ; IN STACK ABLEGEN
    PUSH        H
     .                      ]
     .                      } ; AUSFUEHRUNG DES BEDIENUNGS-
     .                      ; UNTERPROGRAMMS
     .                      ]
    POP         H           ; ALTEN MIKROPROZESSOR-STATUS
    POP         D           ; WIEDERHERSTELLEN
    POP         B
    POP         PSW
    EI                      ; UNTERBRECHUNGS-FREIGABE
    RET                     ; RUECKSPRUNG INS HAUPTPROGRAMM
```

Beachte: Die Reihenfolge der POP-Befehle ist umgekehrt zur PUSH-Reihenfolge!

Prioritäten bei mehreren unterbrechenden externen Geräten: Es sind auch komplexere Unterbrechungs-Vorgänge möglich, bei denen mehrere zur Ausgabe von Unterbrechungsbefehlen berechtigte Peripheriegeräte mit verschiedenen Prioritätsgraden an denselben Prozessor angeschlossen sind. Solche Prioritäten können beim Mikroprozessor 8080 gesetzt werden durch zusätzliche Hardwaremaßnahmen (Logik-Schaltungen in den Interrupt-Leitungen) oder durch Softwaremaßnahmen (z.B. kann innerhalb einer Unterbrechungsroutine der Programmierer durch den Befehl EI dafür sorgen, daß eine Unterbrechung höherer Priorität die erste Unterbrechung ihrerseits unterbrechen kann).

Insgesamt gesehen ist die Möglichkeit von Programmunterbrechungen (Interrupts) eine wichtige Eigenschaft eines Mikrocomputers, die eine optimale Ausnutzung seiner Fähigkeiten erlaubt.

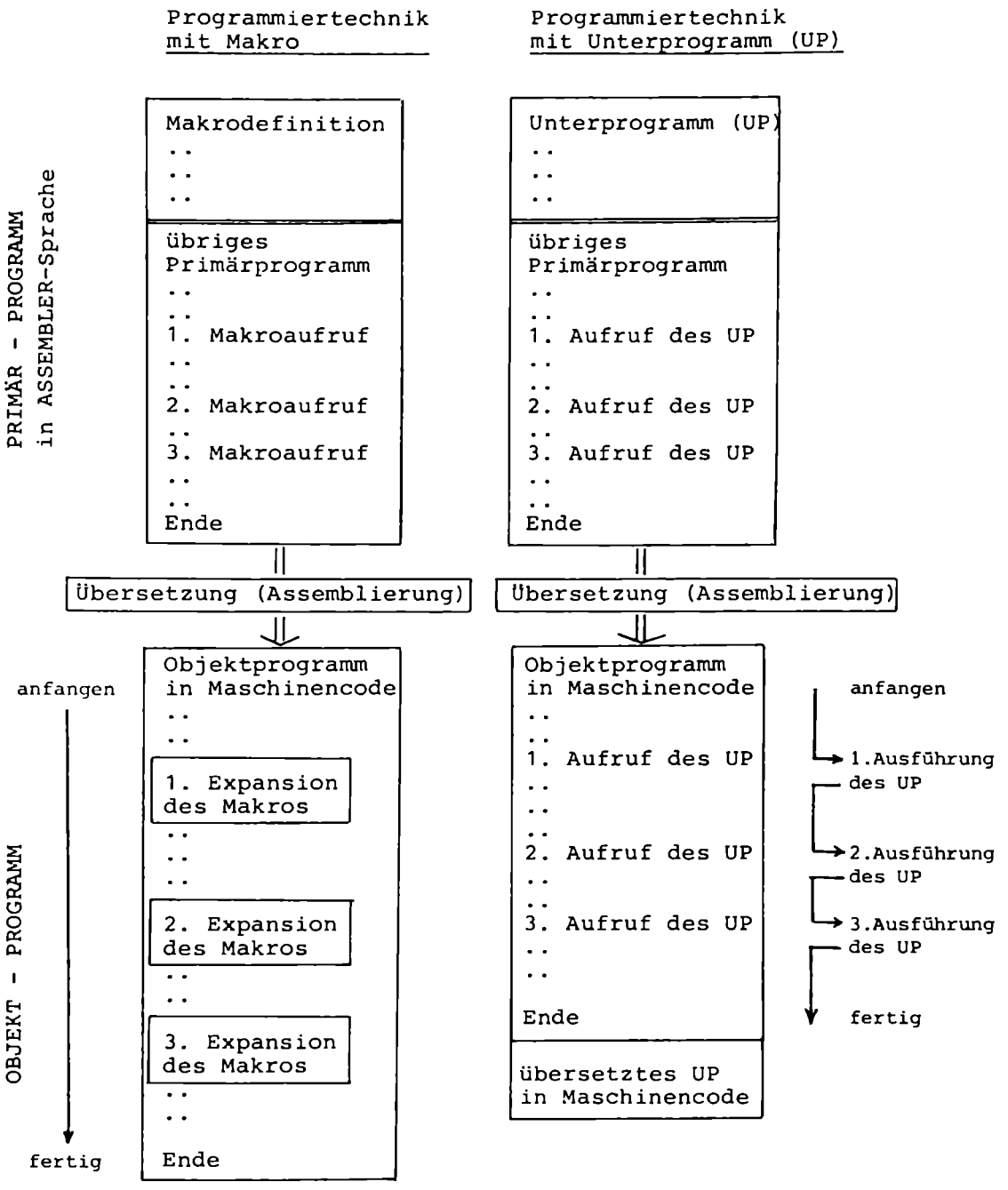
6.7) Makro's

Auf das Programmieren mit Makros soll hier nur kurz eingegangen und das Prinzip erläutert werden.

Wie wir gesehen haben, werden Programmteile, die immer wieder in unveränderter Form benötigt werden ("Routinen"), zweckmäßigerweise nur ein einziges Mal in ASSEMBLER-Sprache erstellt und dann bei Bedarf lediglich in das Arbeitsprogramm des Anwenders eingefügt. Zum Einbau solcher "Programmstücke" in das in Maschinensprache übersetzte Programm ("Objektprogramm") stehen zwei verschiedene Methoden zur Verfügung:

- das Programmstück wird nur einmal als Unterprogramm im Objektprogramm aufgeführt und bei Bedarf durch CALL-Befehle aufgerufen ("Aufruf des Unterprogramms")
- das Programmstück wird jedesmal dort, wo es benötigt wird, in das Objektprogramm vollständig eingebaut ("Expansion des Makro").

Das folgende Schema zeigt die unterschiedliche Behandlung von Makro und Unterprogramm bei der Übersetzung in die Maschinensprache (spielt eine Rolle bei computerunterstützter Übersetzung).



Man erkennt, daß der Makro schon während der Übersetzung ins Objektprogramm bei jedem Makroaufruf eingesetzt wird. Dadurch wird das Anwenderprogramm natürlich um so länger, je öfter darin ein Makroaufruf vorkommt (1., 2., 3., ... "Expansion" des Makros).

Im Gegensatz dazu tritt ein Unterprogramm auch im Maschinencode nur ein einziges Mal in Erscheinung. Es bleibt als ein separater "Programmteil" bestehen, der bei jedem Unterprogramm-Aufruf ausgeführt wird, ohne daß seine Befehle in das Hauptprogramm eingefügt worden sind.

Man sieht also, daß bei der Verwendung von Unterprogrammen weniger Programm-Speicherplatz aufgewendet werden muß, sobald der Programmteil mehr als einmal aufgerufen wird. Da andererseits die Ausführung eines Makros weniger Zeit beansprucht als die Ausführung eines Unterprogrammes, ermöglichen Makros die Durchführung gleicher Aufgaben in kürzerer Zeit, als dies durch Unterprogramm-Aufrufe möglich ist.
Unterprogramme wird man also einsetzen, wenn Routinen sehr häufig verwendet werden oder minimaler Programmspeicherbedarf wesentlich ist. Makros wird man einsetzen, wenn maximale Verarbeitungsgeschwindigkeit gefordert ist und der Programmspeicherbedarf nicht ausschlaggebend ist.

7) DER BEFEHLSSATZ DES MIKROPROZESSORS 8080

Ein Mikroprozessor kann nur solche Befehle ausführen, die ihm in seiner spezifischen Maschinensprache als Bitmuster eingegeben werden und für die er konstruiert ist. Unterschiedliche Mikroprozessortypen haben im allgemeinen unterschiedliche Maschinensprachen, die nicht untereinander austauschbar sind ("nicht kompatibel"). Auch die bequemen mnemonischen Abkürzungen sind von Prozessor zu Prozessor verschieden. Dies ist eine für Mikrocomputer-Anwendungen charakteristische Schwierigkeit beim "Umsteigen" auf einen anderen Mikroprozessor-Typ.

Der Befehlssatz des Mikroprozessors 8080 besteht aus 244 Befehlen (von 2^8 = 256 möglichen). Diese 244 Befehle werden im folgenden kurz erläutert. Die Angabe eines Befehls erfolgt zunächst in der mnemonischen Schreibweise (ASSEMBLER-Sprache). Danach wird der Befehlsinhalt erläutert. Im Anschluß daran erfolgt die Darstellung des Befehls im Maschinencode. Ferner wird bei jedem Befehl angegeben, wieviele Maschinenzyklen und Takte der Mikroprozessor 8080 zur Ausführung des Befehles braucht (die abweichende Zahl von Takten beim Mikroprozessor 8085 steht in Klammer), und es wird angemerkt, welche Bedingungsbits (Flags) bei der Durchführung des Befehls beeinflußt werden können.

Die 244 Befehle des Mikroprozessors 8080 kann man unterteilen in:

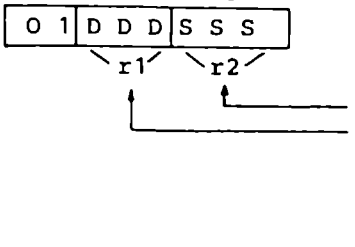
- 96 Befehle für Datenübertragung (Transfer-Befehle): Move, Load/Store, Push/Pop, Input/Output.
- 65 Befehle für arithmetische Operationen: Addition, Subtraktion, Inkrementieren, Dekrementieren.
- 43 Befehle für logische Operationen: UND, ODER, EXCLUSIV-ODER, Komplementieren, Vergleichen (Compare), Rotieren.
- 36 Befehle für Programm-Sprünge: Jump, Call, Return, Restart.
- 4 sonstige Steuerbefehle

Die 244 Befehle des Mikroprozessors 8080 wurden beim Befehlssatz des Mikroprozessors 8085 unverändert übernommen. Auch im Befehlssatz des Mikroprozessors Z80 sind die Befehle des Mikroprozessors 8080 enthalten, jedoch wurden hier die mnemonischen Abkürzungen geändert. Die entsprechenden Mnemonics des Z80 sind auf Seite 196 zusammengestellt, der Maschinencode der Befehle ist identisch.

DATEN-TRANSFER BEFEHLE

MOV r1,r2 (move register to register)

Der Inhalt des Registers r2 (Senderegister) wird ins Register r1 (Zielregister) gebracht.

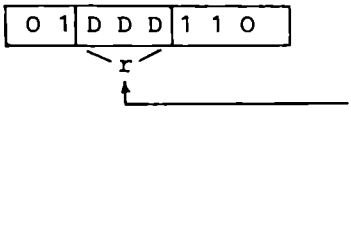


Für das Senderegister r2 (Source) und das Zielregister r1 (Destination) müssen die entsprechenden Bits in den Operationscode eingesetzt werden, z.B. MOV B,A : 0 1 0 0 0 1 1 1

- 1 Zyklus, 5 Takte (4)
- veränderte Flags: keine

MOV r,M (move memory to register)

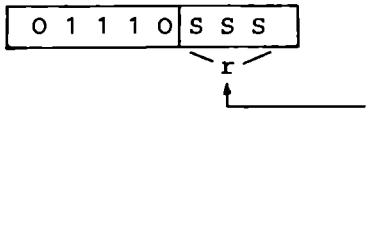
Der Inhalt des Speicherplatzes M, dessen Adresse im Registerpaar H,L steht, wird ins Register r gebracht.



- 2 Zyklen, 7 Takte
- veränderte Flags: keine

MOV M,r (move register to memory)

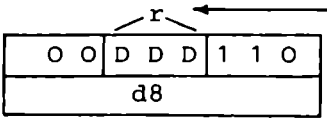
Der Inhalt des Registers r wird an den Speicherplatz M gebracht, dessen Adresse im Registerpaar H,L steht.



- 2 Zyklen, 7 Takte
- veränderte Flags: keine

MVI r,d8 (move immediate to register)

Das 8-Bit-Datenwort d8 wird ins Register r gebracht.

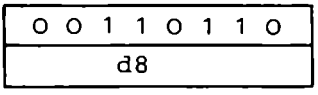


111 für Register A
000 für Register B
001 für Register C
010 für Register D
011 für Register E
100 für Register H
101 für Register L

- 2 Zyklen, 7 Takte
- veränderte Flags: keine

MVI M,d8 (move immediate to memory)

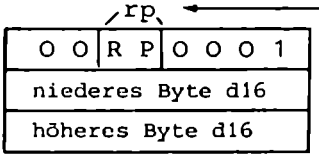
Das 8-Bit-Datenwort d8 wird an den Speicherplatz M gebracht, dessen Adresse im Registerpaar H,L steht.



- 3 Zyklen, 10 Takte
- veränderte Flags: keine

LXI rp,d16 +) (load register pair immediate)

Das 16-Bit-Datenwort d16 wird ins Registerpaar rp gebracht, und zwar das höhere Byte von d16 ins erste Register, das niedere Byte von d16 ins zweite Register des Registerpaares.



00 für Registerpaar B (B und C)
01 für Registerpaar D (D und E)
10 für Registerpaar H (H und L)
11 für Stack Pointer SP

(man beachte die schon erwähnte Eigenart, daß im Maschinencode des Befehls das höhere Byte von d16 nach dem niederen Byte angeordnet ist!)

- 3 Zyklen, 10 Takte
- veränderte Flags: keine

+) Ein "X" in einer Befehlsbezeichnung zeigt an, daß mit einem Registerpaar gearbeitet wird.

LDA adr (load accumulator direct)

Der Inhalt des Speicherplatzes mit der Adresse adr wird in den Akkumulator gebracht.

0 0 1 1 1 0 1 0
niederes Byte Adresse
höheres Byte Adresse

(auch hier steht das höhere Byte der Adresse nach dem niederen Byte im Maschinencode)

- 4 Zyklen, 13 Takte
- veränderte Flags: keine

STA adr (store accumulator direct)

Der Inhalt des Akkumulators wird an den Speicherplatz mit der angegebenen Adresse adr gebracht.

0 0 1 1 0 0 1 0
niederes Byte Adresse
höheres Byte Adresse

- 4 Zyklen, 13 Takte
- veränderte Flags: keine

LHLD adr (load H and L direct)

Der Inhalt des Speicherplatzes mit der angegebenen Adresse adr wird ins Register L gebracht, der Inhalt des darauffolgenden Speicherplatzes wird ins Register H gebracht.

0 0 1 0 1 0 1 0
niederes Byte Adresse
höheres Byte Adresse

- 5 Zyklen, 16 Takte
- veränderte Flags: keine

SHLD adr (store H and L direct)

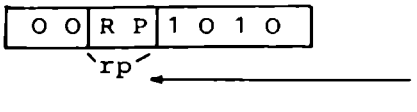
Der Inhalt des Registers L wird an den Speicherplatz mit der angegebenen Adresse adr gebracht. Der Inhalt des Registers H wird an den darauffolgenden Speicherplatz gebracht.

0 0 1 0 0 0 1 0
niederes Byte Adresse
höheres Byte Adresse

- 5 Zyklen, 16 Takte
- veränderte Flags: keine

LDAX rp (load accumulator indirect)

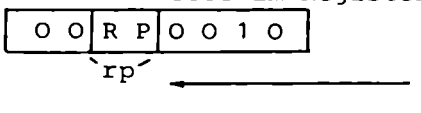
Der Inhalt des Speicherplatzes, dessen Adresse im Registerpaar rp steht, wird in den Akkumulator gebracht.



- 2 Zyklen, 7 Takte
- veränderte Flags: keine

STAX rp (store accumulator indirect)

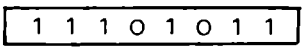
Der Inhalt des Akkumulators wird an den Speicherplatz gebracht, dessen Adresse im Registerpaar rp steht.



- 2 Zyklen, 7 Takte
- veränderte Flags: keine

XCHG (exchange H and L with D and E)

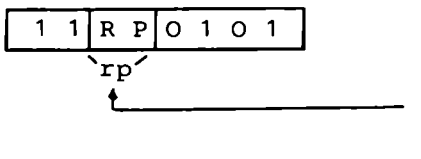
Der Inhalt der Register H und L wird mit dem Inhalt der Register D und E vertauscht.



- 1 Zyklus, 4 Takte
- veränderte Flags: keine

PUSH rp (push register pair on stack)

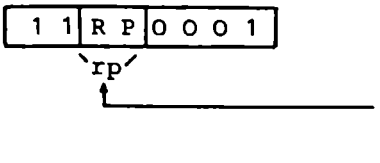
Der Inhalt des Registerpaares rp wird in den STACK gebracht. Und zwar wird der Inhalt des ersten Registers an den Speicherplatz mit der Adresse "Stack Pointer minus 1" gebracht, der Inhalt des zweiten Registers an den Speicherplatz mit der Adresse "Stack Pointer minus 2". (Der Stack Pointer wird anschließend automatisch um 2 erniedrigt).



- 3 Zyklen, 11 Takte (12)
- veränderte Flags: keine

POP rp (pop register pair off stack)

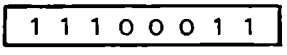
Der Inhalt der beiden "obersten" STACK-Plätze wird in das Registerpaar rp gebracht. Und zwar wird der Inhalt des Speicherplatzes mit der Adresse "Stack Pointer" in das zweite Register gebracht, der Inhalt des Speicherplatzes mit der Adresse "Stack Pointer plus 1" in das erste Register. (Der Stack Pointer wird anschließend automatisch um 2 erhöht).



- 3 Zyklen, 10 Takte
- veränderte Flags: keine, außer beim Befehl POP PSW: alle

XTHL (exchange stack top with H and L)

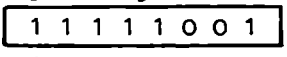
Der Inhalt der beiden "obersten" STACK-Plätze wird mit dem Inhalt des Registerpaares H und L vertauscht. Und zwar wird der Inhalt des Registers L mit dem Inhalt des Speicherplatzes, der durch den Stack-Pointer adressiert wird, vertauscht. Der Inhalt des Registers H wird mit dem Inhalt des Speicherplatzes "Stack Pointer plus 1" vertauscht.



- 5 Zyklen, 18 Takte (16)
- veränderte Flags: keine

SPHL (move H,L to stack pointer SP)

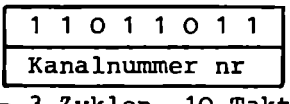
Der Inhalt der Register H und L (16 Bit) wird in das Stack Pointer-Register gebracht.



- 1 Zyklus, 5 Takte (6)
- veränderte Flags: keine

IN nr (input)

Ein 8-Bit-Datenwort wird vom E/A-Kanal mit der Nummer nr in den Akkumulator eingelesen.



- 3 Zyklen, 10 Takte
- veränderte Flags: keine

OUT nr (output)

Ein 8-Bit-Datenwort wird vom Akkumulator an den E/A-Kanal mit der Nummer nr ausgegeben.

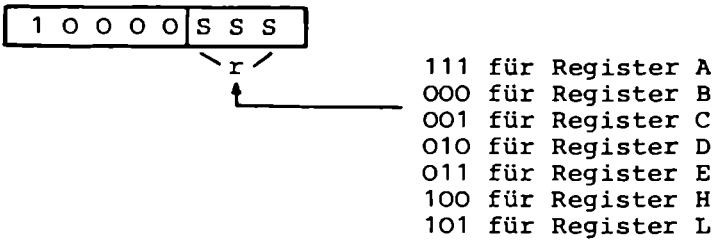
1 1 0 1 0 0 1 1
Kanalnummer nr

- 3 Zyklen, 10 Takte
- veränderte Flags: keine

BEFEHLE FÜR ARITHMETISCHE OPERATIONEN

ADD r (add register to accumulator)

Der Inhalt des Registers r wird zum Inhalt des Akkumulators addiert. Das Ergebnis steht im Akkumulator.



- 1 Zyklus, 4 Takte
- veränderte Flags: alle

ADD M (add memory to accumulator)

Der Inhalt des Speicherplatzes M, dessen Adresse im Registerpaar H,L steht, wird zum Inhalt des Akkumulators addiert. Das Ergebnis steht im Akkumulator.

1 0 0 0 0 1 1 0

- 2 Zyklen, 7 Takte
- veränderte Flags: alle

ADI d8 (add immediate to accumulator)

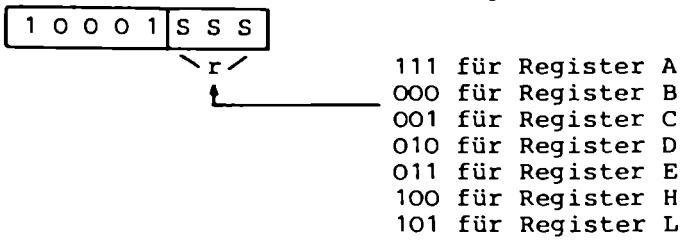
Das 8-Bit-Datenwort d8 wird zum Inhalt des Akkumulators addiert. Das Ergebnis steht im Akkumulator.

1 1 0 0 0 1 1 0
d8

- 2 Zyklen, 7 Takte
- veränderte Flags: alle

ADC r (add register to accumulator with carry)

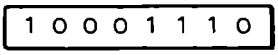
Der Inhalt des Registers r plus das Carry-Bit werden zum Inhalt des Akkumulators addiert. Das Ergebnis steht im Akkumulator.



- 1 Zyklus, 4 Takte
- veränderte Flags: alle

ADC M (add memory to accumulator with carry)

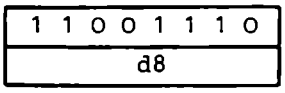
Der Inhalt des Speicherplatzes M, dessen Adresse im Registerpaar H,L steht, plus das Carry-Bit werden zum Inhalt des Akkumulators addiert. Das Ergebnis steht im Akkumulator.



- 2 Zyklen, 7 Takte
- veränderte Flags: alle

ACI d8 (add immediate to accumulator with carry)

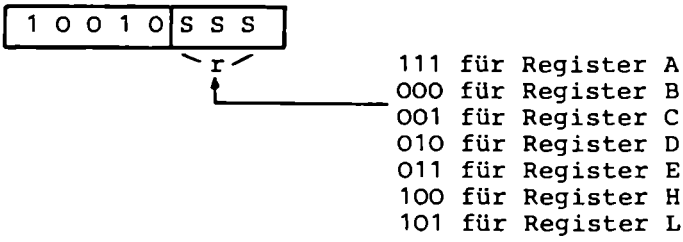
Das 8-Bit-Datenwort d8 plus das Carry-Bit werden zum Inhalt des Akkumulators addiert. Das Ergebnis steht im Akkumulator.



- 2 Zyklen, 7 Takte
- veränderte Flags: alle

SUB r (subtract register from accumulator)

Der Inhalt des Registers r wird vom Inhalt des Akkumulators subtrahiert. Das Ergebnis steht im Akkumulator.



- 1 Zyklus, 4 Takte
- veränderte Flags: alle

SUB M (subtract memory from accumulator)

Der Inhalt des Speicherplatzes M, dessen Adresse im Registerpaar H,L steht, wird vom Inhalt des Akkumulators subtrahiert. Das Ergebnis steht im Akkumulator.

1	0	0	1	0	1	1	0

- 2 Zyklen, 7 Takte
- veränderte Flags: alle

SUI d8 (subtract immediate from accumulator)

Das 8-Bit-Datenwort d8 wird vom Inhalt des Akkumulators subtrahiert. Das Ergebnis steht im Akkumulator.

1	1	0	1	0	1	1	0
d8							

- 2 Zyklen, 7 Takte
- veränderte Flags: alle

SBB r (subtract register from accumulator with borrow)

Der Inhalt des Registers r plus das Carry-Bit werden vom Inhalt des Akkumulators subtrahiert. Das Ergebnis steht im Akkumulator. (Dieser Befehl berücksichtigt, ob eine vorangegangene Subtraktion ein negatives Resultat ("borgen") erzeugt hatte.)

1	0	0	1	1	S	S	S

r

111 für Register A
000 für Register B
001 für Register C
010 für Register D
011 für Register E
100 für Register H
101 für Register L

- 1 Zyklus, 4 Takte
- veränderte Flags: alle

SBB M (subtract memory from accumulator with borrow)

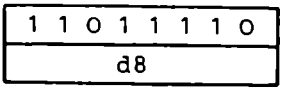
Der Inhalt des Speicherplatzes M, dessen Adresse im Registerpaar H,L steht, plus das Carry-Bit werden vom Inhalt des Akkumulators subtrahiert. Das Ergebnis steht im Akkumulator.

1	0	0	1	1	1	1	0

- 2 Zyklen, 7 Takte
- veränderte Flags: alle

SBI d8 (subtract immediate from accumulator with borrow)

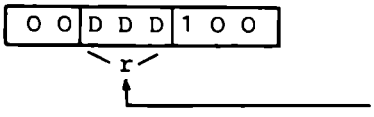
Das 8-Bit-Datenwort d8 plus das Carry-Bit werden vom Inhalt des Akkumulators subtrahiert. Das Ergebnis steht im Akkumulator.



- 2 Zyklen, 7 Takte
- veränderte Flags: alle

INR r (increment register)

Der Inhalt des Registers r wird um 1 erhöht.

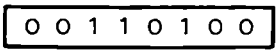


111 für Register A
000 für Register B
001 für Register C
010 für Register D
011 für Register E
100 für Register H
101 für Register L

- 1 Zyklus, 5 Takte (4)
- veränderte Flags: alle außer Carry

INR M (increment memory)

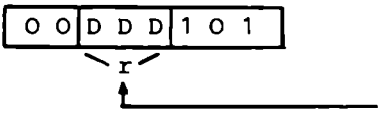
Der Inhalt des Speicherplatzes M, dessen Adresse im Registerpaar H,L steht, wird um 1 erhöht.



- 3 Zyklen, 10 Takte
- veränderte Flags: alle außer Carry

DCR r (decrement register)

Der Inhalt des Registers r wird um 1 erniedrigt.



111 für Register A
000 für Register B
001 für Register C
010 für Register D
011 für Register E
100 für Register H
101 für Register L

- 1 Zyklus, 5 Takte (4)
- veränderte Flags: alle außer Carry

DCR M (decrement memory)

Der Inhalt des Speicherplatzes M, dessen Adresse im Registerpaar H,L steht, wird um 1 erniedrigt.

0 0 1 1 0 1 0 1

- 3 Zyklen, 10 Takte
- veränderte Flags: alle außer Carry

INX rp (increment register pair)

Der Inhalt (16-Bit-Zahl) des Registerpaares rp wird um 1 erhöht.

0 0	R P	0 0 1 1

`rp´

00 für Registerpaar B (B und C)
01 für Registerpaar D (D und E)
10 für Registerpaar H (H und L)
11 für Stack Pointer SP

- 1 Zyklus, 5 Takte (6)
- veränderte Flags: keine

DCX rp (decrement register pair)

Der Inhalt (16-Bit-Zahl) des Registerpaares rp wird um 1 erniedrigt.

0 0	R P	1 0 1 1

`rp´

00 für Registerpaar B (B und C)
01 für Registerpaar D (D und E)
10 für Registerpaar H (H und L)
11 für Stack Pointer SP

- 1 Zyklus, 5 Takte (6)
- veränderte Flags: keine

DAD rp ("double-add" register pair to H and L)

Die 16-Bit-Zahl im Registerpaar rp wird zur 16-Bit-Zahl im Registerpaar H,L addiert. Das Ergebnis steht im Registerpaar H,L.

0 0	R P	1 0 0 1

`rp´

00 für Registerpaar B (B und C)
01 für Registerpaar D (D und E)
10 für Registerpaar H (H und L)
11 für Stack Pointer SP

- 3 Zyklen, 10 Takte
- veränderte Flags: nur Carry

DAA (decimal adjust accumulator)

Die 8-Bit-Zahl im Akkumulator wird in 2 BCD-Ziffern umgewandelt. (Verwendet zur Addition von Dezimalzahlen).

0 0 1 0 0 1 1 1

- 1 Zyklus, 4 Takte
- veränderte Flags: alle

BEFEHLE FÜR LOGISCHE OPERATIONEN

ANA r (AND register with accumulator)

Der Inhalt des Registers r wird mit dem Inhalt des Akkumulators bitweise logisch UNDiert. Das Ergebnis steht im Akkumulator. Das Carry-Bit wird rückgesetzt.

1 0 1 0 0	S S S

r

111 für Register A
000 für Register B
001 für Register C
010 für Register D
011 für Register E
100 für Register H
101 für Register L

- 1 Zyklus, 4 Takte
- veränderte Flags: alle

ANA M (AND memory with accumulator)

Der Inhalt des Speicherplatzes M, dessen Adresse im Registerpaar H,L steht, wird mit dem Inhalt des Akkumulators bitweise UNDiert. Das Ergebnis steht im Akkumulator. Das Carry-Bit wird rückgesetzt.

1 0 1 0 0 1 1 0

- 2 Zyklen, 7 Takte
- veränderte Flags: alle

ANI d8 (AND immediate with accumulator)

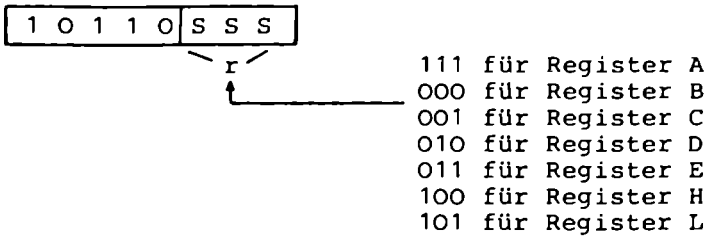
Das 8-Bit-Datenwort d8 wird mit dem Inhalt des Akkumulators bitweise UNDiert. Das Ergebnis steht im Akkumulator. Das Carry-Bit wird rückgesetzt.

1 1 1 0 0 1 1 0
d8

- 2 Zyklen, 7 Takte
- veränderte Flags: alle

ORA r (OR register with accumulator)

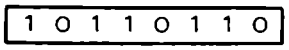
Der Inhalt des Registers r wird mit dem Inhalt des Akkumulators bitweise logisch ODERiert. Das Ergebnis steht im Akkumulator. Das Carry-Bit wird rückgesetzt.



- 1 Zyklus, 4 Takte
- veränderte Flags: alle

ORA M (OR memory with accumulator)

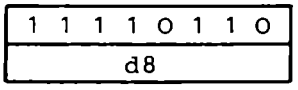
Der Inhalt des Speicherplatzes M, dessen Adresse im Registerpaar H,L steht, wird mit dem Inhalt des Akkumulators bitweise ODERiert. Das Ergebnis steht im Akkumulator. Das Carry-Bit wird rückgesetzt.



- 2 Zyklen, 7 Takte
- veränderte Flags: alle

ORI d8 (OR immediate with accumulator)

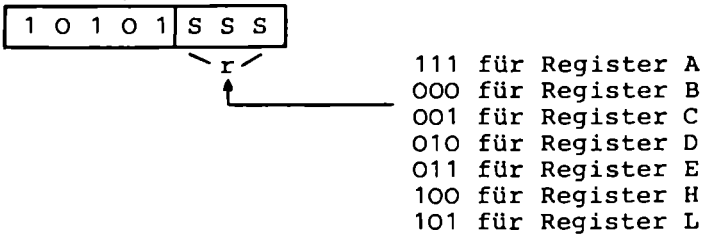
Das 8-Bit-Datenwort d8 wird mit dem Inhalt des Akkumulators bitweise ODERiert. Das Ergebnis steht im Akkumulator. Das Carry-Bit wird rückgesetzt.



- 2 Zyklen, 7 Takte
- veränderte Flags: alle

XRA r (EXCLUSIVE-OR register with accumulator)

Der Inhalt des Registers r wird mit dem Inhalt des Akkumulators bitweise logisch EXCLUSIV-ODERiert. Das Ergebnis steht im Akkumulator. Das Carry-Bit wird rückgesetzt.



- 1 Zyklus, 4 Takte
- veränderte Flags: alle

XRA M (EXCLUSIVE-OR memory with accumulator)

Der Inhalt des Speicherplatzes M, dessen Adresse im Registerpaar H,L steht, wird mit dem Inhalt des Akkumulators bitweise EXCLUSIV-ODERiert. Das Ergebnis steht im Akkumulator. Das Carry-Bit wird rückgesetzt.

1 0 1 0 1 1 1 0

- 2 Zyklen, 7 Takte
- veränderte Flags: alle

XRI d8 (EXCLUSIVE-OR immediate with accumulator)

Das 8-Bit-Datenwort d8 wird mit dem Inhalt des Akkumulators bitweise EXCLUSIV-ODERiert. Das Ergebnis steht im Akkumulator. Das Carry-Bit wird rückgesetzt.

1 1 1 0 1 1 1 0
d8

- 2 Zyklen, 7 Takte
- veränderte Flags: alle

CMP r (compare register with accumulator)

Der Inhalt des Registers r wird mit dem Inhalt des Akkumulators intern verglichen+): Das Zero-Bit wird gesetzt, wenn die Inhalte gleich sind, sonst rückgesetzt. Das Carry-Bit wird gesetzt, wenn der Inhalt des Registers r größer ist als der Inhalt des Akkumulators, sonst rückgesetzt. Register r und Akkumulator bleiben unverändert.

1 0 1 1 1	S S S

r

111 für Register A
000 für Register B
001 für Register C
010 für Register D
011 für Register E
100 für Register H
101 für Register L

- 1 Zyklus, 4 Takte
- veränderte Flags: alle

CMP M (compare memory with accumulator)

Der Inhalt des Speicherplatzes M, dessen Adresse im Register H,L steht, wird mit dem Inhalt des Akkumulators intern verglichen+): Das Zero-Bit wird gesetzt, wenn die Inhalte gleich sind, sonst rückgesetzt. Das Carry-Bit wird gesetzt, wenn der Inhalt des Speicher-

---

+) Der Vergleich erfolgt durch Subtraktion vom Akkumulator

platzes M größer ist als der Inhalt des Akkumulators, sonst rückgesetzt. Speicherplatz M und Akkumulator bleiben unverändert.

1	0	1	1	1	1	1	0

- 2 Zyklen, 7 Takte
- veränderte Flags: alle

CPI d8 (compare immediate with accumulator)

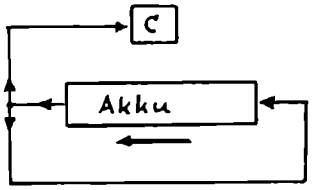
Das 8-Bit-Datenwort d8 wird mit dem Inhalt des Akkumulators intern verglichen*): Das Zero-Bit wird gesetzt, wenn der Inhalt des Akkumulators gleich d8 ist, sonst rückgesetzt. Das Carry-Bit wird gesetzt, wenn d8 größer ist als der Inhalt des Akkumulators, sonst rückgesetzt. Der Akkumulator bleibt unverändert.

1	1	1	1	1	1	1	0
d8							

- 2 Zyklen, 7 Takte
- veränderte Flags: alle

RLC (rotate accumulator left circular)

Der Inhalt des Akkumulators wird zyklisch um 1 Stelle nach links verschoben. Das niederste Bit und das Carry-Bit werden gleich dem hinausgeschobenen Bit gesetzt.

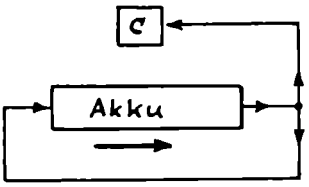


0	0	0	0	0	1	1	1

- 1 Zyklus, 4 Takte
- veränderte Flags: nur Carry

RRC (rotate accumulator right circular)

Der Inhalt des Akkumulators wird zyklisch um 1 Stelle nach rechts verschoben. Das höchste Bit und das Carry-Bit werden gleich dem hinausgeschobenen Bit gesetzt.

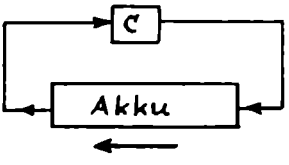


0	0	0	0	1	1	1	1

- 1 Zyklus, 4 Takte
- veränderte Flags: nur Carry

RAL (rotate accumulator left through carry)

Der Inhalt des Akkumulators wird zyklisch um 1 Stelle nach links "durch das Carry-Bit hindurch" verschoben: Das niederste Bit wird durch das Carry-Bit ersetzt und das Carry-Bit durch das hinausgeschobene Bit ersetzt.

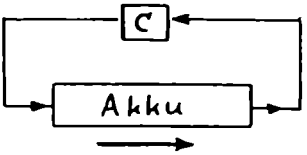


0	0	0	1	0	1	1	1

- 1 Zyklus, 4 Takte
- veränderte Flags: nur Carry

RAR (rotate accumulator right through carry)

Der Inhalt des Akkumulators wird zyklisch um 1 Stelle nach rechts "durch das Carry-Bit hindurch" verschoben: Das höchste Bit wird durch das Carry-Bit ersetzt und das Carry-Bit durch das hinausgeschobene Bit ersetzt.



0 0 0 1 1 1 1 1

- 1 Zyklus, 4 Takte
- veränderte Flags: nur Carry

CMA (complement accumulator)

Der Inhalt des Akkumulators wird bitweise invertiert (0 wird 1, 1 wird 0).

0 0 1 0 1 1 1 1

- 1 Zyklus, 4 Takte
- veränderte Flags: keine

CMC (complement carry)

Das Carry-Bit wird invertiert.

0 0 1 1 1 1 1 1

- 1 Zyklus, 4 Takte
- veränderte Flags: nur Carry

STC (set carry)

Das Carry-Bit wird gleich 1 gesetzt.

0 0 1 1 0 1 1 1

- 1 Zyklus, 4 Takte
- veränderte Flags: nur Carry

BEFEHLE FÜR PROGRAMMSPRÜNGE

JMP adr (jump unconditional)

Das Programm wird bei der Adresse adr fortgesetzt (unbedingt).

1 1 0 0 0 0 1 1
niederes Byte Adresse
höheres Byte Adresse

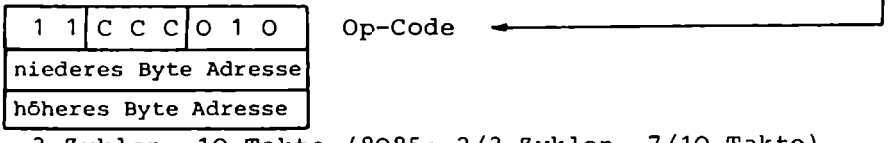
- 3 Zyklen, 10 Takte
- veränderte Flags: keine

Bedingte JUMP-Befehle:

Das Programm wird an der Adresse adr fortgesetzt, wenn

- das Zero-Bit = 0 ist:	JNZ adr (jump if not zero)	1 1 0 0 0 0 1 0 Op-Code
- das Zero-Bit = 1 ist:	JZ adr (jump if zero)	1 1 0 0 1 0 1 0 Op-Code
- das Carry-Bit = 0 ist:	JNC adr (jump if no carry)	1 1 0 1 0 0 1 0 Op-Code
- das Carry-Bit = 1 ist:	JC adr (jump if carry)	1 1 0 1 1 0 1 0 Op-Code
- das Parity-Bit = 0 ist:	JPO adr (jump if parity odd)	1 1 1 0 0 0 1 0 Op-Code
- das Parity-Bit = 1 ist:	JPE adr (jump if parity even)	1 1 1 0 1 0 1 0 Op-Code
- das Sign-Bit = 0 ist:	JP adr (jump if plus)	1 1 1 1 0 0 1 0 Op-Code
- das Sign-Bit = 1 ist:	JM adr (jump if minus)	1 1 1 1 1 0 1 o Op-Code

Das Format dieser 8 Befehle lautet:



- 3 Zyklen, 10 Takte (8085: 2/3 Zyklen, 7/10 Takte)
- veränderte Flags: keine

CALL adr (call unconditional)

Das Programm springt in ein Unterprogramm mit der Anfangsadresse adr (unbedingt).

1 1 0 0 1 1 0 1
niederes Byte Adresse
höheres Byte Adresse

- 5 Zyklen, 17 Takte (18)
- veränderte Flags: keine

Bedingte CALL-Befehle:

Das Programm springt in ein Unterprogramm mit der Anfangsadresse adr, wenn

- das Zero-Bit = 0 ist:	CNZ adr (call if not zero)	1 1 0 0 0 1 0 0 Op-Code
- das Zero-Bit = 1 ist:	CZ adr (call if zero)	1 1 0 0 1 1 0 0 Op-Code
- das Carry-Bit = 0 ist:	CNC adr (call if no carry)	1 1 0 1 0 1 0 0 Op-Code
- das Carry-Bit = 1 ist:	CC adr (call if carry)	1 1 0 1 1 1 0 0 Op-Code
- das Parity-Bit = 0 ist:	CPO adr (call if parity odd)	1 1 1 0 0 1 0 0 Op-Code
- das Parity-Bit = 1 ist:	CPE adr (call if parity even)	1 1 1 0 1 1 0 0 Op-Code
- das Sign-Bit = 0 ist:	CP adr (call if plus)	1 1 1 1 0 1 0 0 Op-Code
- das Sign-Bit = 1 ist:	CM adr (call if minus)	1 1 1 1 1 1 0 0 Op-Code

Das Format dieser 8 Befehle lautet:

1 1 | C C C | 1 0 0	Op-Code ←
niederes Byte Adresse	
höheres Byte Adresse	

- 3/5 Zyklen, 11/17 Takte (8085: 2/5 Zyklen, 9/18 Takte)
- veränderte Flags: keine

RET (return unconditional)

Es wird aus dem Unterprogramm ins verlassene Programm zurückgesprungen (unbedingt). (Die Rücksprungadresse wird automatisch aus dem STACK geholt.)

1	1	0	0	1	0	0	1

- 3 Zyklen, 10 Takte
- veränderte Flags: keine

Bedingte RETURN-Befehle:

Es wird aus dem Unterprogramm ins verlassene Programm zurückgesprungen, wenn

- das Zero-Bit = 0 ist: RNZ (return if not zero)

1	1	0	0	0	0	0	0

- das Zero-Bit = 1 ist: RZ (return if zero)

1	1	0	0	1	0	0	0

- das Carry-Bit = 0 ist: RNC (return if no carry)

1	1	0	1	0	0	0	0

- das Carry-Bit = 1 ist: RC (return if carry)

1	1	0	1	1	0	0	0

- das Parity-Bit = 0 ist: RPO (return if parity odd)

1	1	1	0	0	0	0	0

- das Parity-Bit = 1 ist: RPE (return if parity even)

1	1	1	0	1	0	0	0

- das Sign-Bit = 0 ist: RP (return if plus)

1	1	1	1	0	0	0	0

- das Sign-Bit = 1 ist: RM (return if minus)

1	1	1	1	1	0	0	0

Das Format dieser 8 Befehle lautet:

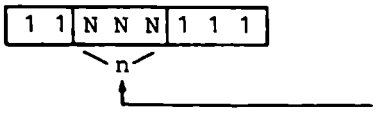
1	1	C	C	C	0	0	0

- 1/3 Zyklen, 5/11 Takte (6/12)
- veränderte Flags: keine

RST n (restart)

Das Programm springt auf die Adresse 8·n (n = 0,1,2,...7).

Diese speziellen Unterprogramm-Aufrufbefehle werden im allgemeinen nur von einem peripheren Gerät bei Anforderung von Interrupt-Service-Routinen benützt (siehe Kapitel 6). Sie kommen seltener als Software-Befehle in einem Programm vor.

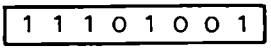


000	für RST 0 --	Adresse 0	(0000 H)
001	für RST 1 --	Adresse 8	(0008 H)
010	für RST 2 --	Adresse 16	(0010 H)
011	für RST 3 --	Adresse 24	(0018 H)
100	für RST 4 --	Adresse 32	(0020 H)
101	für RST 5 --	Adresse 40	(0028 H)
110	für RST 6 --	Adresse 48	(0030 H)
111	für RST 7 --	Adresse 56	(0038 H)

- 3 Zyklen, 11 Takte (12)
- veränderte Flags: keine

PCHL · (move H and L to program-counter PC)

Der Inhalt des Befehlszählers PC wird durch den Inhalt des Registerpaares H,L ersetzt. D.h. das Programm springt auf die Adresse, die im Registerpaar H,L steht.

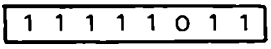


- 1 Zyklus, 5 Takte (6)
- veränderte Flags: keine

SONSTIGE STEUERBEFEHLE

EI (enable interrupt)

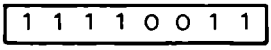
Interrupt-Anforderungen werden vom Mikroprozessor angenommen. (Das INTE-Flipflop wird gesetzt.)



- 1 Zyklus, 4 Takte
- veränderte Flags: keine

DI (disable interrupt)

Interrupt-Anforderungen werden vom Mikroprozessor ignoriert. (Das INTE-Flipflop wird rückgesetzt.)



- 1 Zyklus, 4 Takte
- veränderte Flags: keine

HLT (halt)

Der Mikroprozessor wird gestoppt, bis eine Interrupt-Anforderung erfolgt. (Oder der Prozessor durch RESET auf Null zurückgesetzt wird)

0	1	1	1	0	1	1	0

- 1 Zyklus, 7 Takte (5)
- veränderte Flags: keine

NOP (no operation)

Leerbefehl

0	0	0	0	0	0	0	0

- 1 Zyklus, 4 Takte
- veränderte Flags: keine

Auf der folgenden Seite sind sämtliche Befehle der Mikroprozessoren 8080/8085 übersichtlich nach Befehlsgruppen geordnet zusammengestellt. Eine Auflistung der 8080/85-Befehle in der hexadezimalen Reihenfolge der Opcodes wurde schon auf Seite 16 gegeben.

Transfer 8 Bit

INPUT/OUTPUT: D3 OUT / DB IN } Nr

MOVE IMMEDIATE

06 MVI B,		
0E MVI C,		
16 MVI D,		
1E MVI E,	D8	
26 MVI H,		
2E MVI L,		
36 MVI M,		
3E MVI A,		

MOVE

	40 MOV B,B	50 MOV D,B	60 MOV H,B	70 MOV M,B
	41 MOV B,C	51 MOV D,C	61 MOV H,C	71 MOV M,C
	42 MOV B,D	52 MOV D,D	62 MOV H,D	72 MOV M,D
	43 MOV B,E	53 MOV D,E	63 MOV H,E	73 MOV M,E
	44 MOV B,H	54 MOV D,H	64 MOV H,H	74 MOV M,H
	45 MOV B,L	55 MOV D,L	65 MOV H,L	75 MOV M,L
	46 MOV B,M	56 MOV D,M	66 MOV H,M	—
	47 MOV B,A	57 MOV D,A	67 MOV H,A	77 MOV M,A
	48 MOV C,B	58 MOV E,B	68 MOV L,B	78 MOV A,B
	49 MOV C,C	59 MOV E,C	69 MOV L,C	79 MOV A,C
	4A MOV C,D	5A MOV E,D	6A MOV L,D	7A MOV A,D
	4B MOV C,E	5B MOV E,E	6B MOV L,E	7B MOV A,E
	4C MOV C,H	5C MOV E,H	6C MOV L,H	7C MOV A,H
	4D MOV C,L	5D MOV E,L	6D MOV L,L	7D MOV A,L
	4E MOV C,M	5E MOV E,M	6E MOV L,M	7E MOV A,M
	4F MOV C,A	5F MOV E,A	6F MOV L,A	7F MOV A,A

LOAD/STORE

3A LDA		
32 STA	Adr	
0A LDAX B		
1A LDAX D		
02 STAX B		
12 STAX D		

Transfer 16 Bit

LOAD IMMEDIATE

01 LXI B,		
11 LXI D,	D16	
21 LXI H,		
31 LXI SP,		

LOAD/STORE

2A LHLD	Adr
22 SHLD	
EB XCHG	

STACK

C5 PUSH B	C1 POP B
D5 PUSH D	D1 POP D
E5 PUSH H	E1 POP H
F5 PUSH PSW	F1 POP PSW *
E3 XTHL	
F9 SPHL	

Arithmetische Operation 8 Bit

ADDITION *	SUBTRACTION *	IN-CREMENT (*)	DE-CREMENT (*)
80 ADD B	88 ADC B	90 SUB B	98 SBB B	04 INR B	05 DCR B
81 ADD C	89 ADC C	91 SUB C	99 SBB C	0C INR C	0D DCR C
82 ADD D	8A ADC D	92 SUB D	9A SBB D	14 INR D	15 DCR D
83 ADD E	8B ADC E	93 SUB E	9B SBB E	1C INR E	1D DCR E
84 ADD H	8C ADC H	94 SUB H	9C SBB H	24 INR H	25 DCR H
85 ADD L	8D ADC L	95 SUB L	9D SBB L	2C INR L	2D DCR L
86 ADD M	8E ADC M	96 SUB M	9E SBB M	34 INR M	35 DCR M
87 ADD A	8F ADC A	97 SUB A	9F SBB A	3C INR A	3D DCR A
C6 ADI D8	CE ACI D8	D6 SUI D8	DE SBI D8		

Arithmetische Operation 16 Bit

ADDITION †	INCREMENT	DECREMENT
09 DAD B	03 INX B	0B DCX B
19 DAD D	13 INX D	1B DCX D
29 DAD H	23 INX H	2B DCX H
39 DAD SP	33 INX SP	3B DCX SP

DECIMAL ADJUST *

27 DAA

Logische Operation 8 Bit

AND *	X-OR *	OR *	COMPARE *	ROTATE †
				07 RLC
A0 ANA B	A8 XRA B	B0 ORA B	B8 CMP B	0F RRC
A1 ANA C	A9 XRA C	B1 ORA C	B9 CMP C	17 RAL
A2 ANA D	AA XRA D	B2 ORA D	BA CMP D	1F RAR
A3 ANA E	AB XRA E	B3 ORA E	BB CMP E	
A4 ANA H	AC XRA H	B4 ORA H	BC CMP H	COMPLEMENT
A5 ANA L	AD XRA L	B5 ORA L	BD CMP L	2F CMA
A6 ANA M	AE XRA M	B6 ORA M	BE CMP M	37 STC †
A7 ANA A	AF XRA A	B7 ORA A	BF CMP A	3F CMC †
E6 ANI D8	EE XRI D8	F6 ORI D8	FE CPI D8	

Sprung

JUMP		CALL		RETURN	RESTART
C3 JMP		CD CALL		C9 RET	C7 RST 0
C2 JNZ		C4 CNZ		C0 RNZ	CF RST 1
CA JZ		CC CZ		C8 RZ	D7 RST 2
D2 JNC		D4 CNC		D0 RNC	DF RST 3
DA JC		DC CC		D8 RC	E7 RST 4
E2 JPO	Adr	E4 CPO	Adr	E0 RPO	EF RST 5
EA JPE		EC CPE		E8 RPE	F7 RST 6
F2 JP		F4 CP		F0 RP	FF RST 7
FA JM		FC CM		F8 RM	
E9 PCHL					

Sonstige

CONTROL		
00 NOP		
76 HLT	20	RIM
F3 DI	30	SIM
FB EI		

* alle Flags beeinflußt

(*) alle Flags außer Carry beeinflußt

† nur Carry beeinflußt

Der Befehlssatz des Mikroprozessors 8080/85

nach Befehlsgruppen geordnet

(mit hexadezimaler Darstellung des Operations-Codes der Befehle)

(RIM und SIM sind nur beim 8085 anwendbar)

8) ERSTELLEN VON PROGRAMMEN FÜR DEN MIKROCOMPUTER 8080

Der Mikroprozessor 8080 arbeitet - wie jeder andere Zentralprozessor - in seiner eigenen spezifischen Maschinensprache. Wie bereits erwähnt, ist die Form dieser Maschinensprache für die menschliche Denkweise sehr unbequem: Die Befehle bestehen aus einer schwer zu überblickenden Folge von 8 bis 24 Einsen oder Nullen. Man hat daher eine für den Anwender bequemere Sprache mit mnemonischen Abkürzungen der Befehle entwickelt, die ASSEMBLER-Sprache. Trotzdem muß natürlich letztendlich der Mikrocomputer mit den für ihn spezifischen Bitmustern aus Einsen und Nullen gefüttert werden. Das heißt, der Anwender, der ein Programm für einen Mikrocomputer zu entwickeln hat, erstellt dieses zunächst in der bequemen ASSEMBLER-Sprache: Das so entstandene "Primärprogramm oder Quellenprogramm" in ASSEMBLER-Sprache muß dann in die Maschinensprache übersetzt werden und ergibt das "Objektprogramm" in Maschinensprache, das dann in den Speicher des Mikrocomputers eingegeben wird. Das Übersetzen eines Programms aus der ASSEMBLER-Sprache in die Maschinensprache nennt man "Assemblierung". Für kleinere Programme kann man diese Assemblierung "von Hand" durchführen, was eine schematische und sehr mühsame Arbeit ist. Bei größeren Programmen läßt man die Übersetzung (Assemblierung) durch einen Computer (Minicomputer, Mikrocomputer-Entwicklungssystem) durchführen, wozu der Computer ein spezielles Übersetzungsprogramm - das "Assembler-Programm"- benötigt. Dieses Assembler-Programm ist dann natürlich auch von Prozessortyp zu Prozessortyp verschieden. Bei Benützung des Wortes "Assembler" ist also zu unterscheiden, ob man die ASSEMBLER-Sprache oder das Assembler-Übersetzungsprogramm - "den Assembler" (= Übersetzer) - meint.

8.1) Übersetzen (Assemblierung) "von Hand"

Kleinere Programme für den Mikrocomputer 8080 kann man von Hand erstellen, d.h. ohne Zuhilfenahme eines Computers für die Übersetzung des Primärprogramms in die Maschinensprache. Man wird

aber auch diese kleineren Programme zunächst in ASSEMBLER-Sprache erstellen und sie dann Zeile für Zeile von Hand in die Maschinensprache übertragen, wobei man die übersichtlichere Hexadezimal-Darstellung der Maschinensprache benutzt.

8.11) Erstes Programmbeispiel:

Aufgabe: Es sollen zwei Zahlen zwischen 0 und 255 addiert werden. Die Zahlen sollen sich an zwei Speicherplätzen "Adr 1" und "Adr 2" des RAM-Speichers befinden. Das Resultat soll wieder im RAM auf dem Speicherplatz "Adr 1" abgelegt werden.

- Die erste Zahl sei 1A, sie befinde sich auf der RAM-Adresse "Adr 1"= OEO1.
- Die zweite Zahl sei B6, sie befinde sich auf der RAM-Adresse "Adr 2"= A12D.

Der Programmablauf gestaltet sich nun wie folgt: Zur Addition muß eine der beiden Zahlen - hier B6 - in den Akkumulator gebracht werden: LDA A12D. Die Addition der anderen Zahl aus dem Speicher kann nun durch den Befehl ADD M erfolgen, d.h. durch indirekte Adressierung per Registerpaar H,L. Die Adresse "Adr 1" muß also zuvor ins Registerpaar H,L gebracht werden: LXI H,OEO1. Das Resultat der Addition steht zunächst im Akkumulator, es muß dann noch auf den RAM-Platz "Adr 1" transferiert werden durch den indirekt adressierenden Befehl MOV M,A. Das Ergebnis der Addition 1A + B6 = DO (oder dezimal: 26 + 182 = 208) steht also im Speicherplatz OEO1. Ein evtl. vorkommender Überlauf bei der Addition würde im Carrybit angezeigt werden.

Die folgende Tabelle zeigt in der 3. Spalte das primäre Quellenprogramm in ASSEMBLER-Sprache, in der 2. Spalte die "von Hand" in den Maschinencode übersetzten Befehle und in der 1. Spalte die Adressen der aufeinanderfolgenden 8 Speicherplätze, in die das Programm eingeschrieben wurde (das "Additionsprogramm" beginnt also im Speicher bei der Adresse 0020). Auf der rechten Seite der Tabelle werden in den Spalten 4 - 7 die Veränderungen in den Registern und im RAM-Speicher aufgeführt, die sich innerhalb des Mikrocomputers während der Ausführung der Befehle ergeben.

Programmspeicher Adresse (hex)	Inhalt (hex)	Primärprogramm in ASSEMBLER-Sprache	Auswirkungen der Befehle auf Register und Speicherplätze Inhalt Akku	Inhalt H,L	Inhalt RAM OEO1	Inhalt RAM A12D
			?	?	1A	B6
0020	3A					
0021	2D	LDA A12D				
0022	A1					
			B6	?	1A	B6
0023	21					
0024	01	LXI H,OEO1				
0025	OE					
			B6	OEO1	1A	B6
0026	86	ADD M				
			DO	OEO1	1A	B6
0027	77	MOV M,A				
			DO	OEO1	DO	B6

Die Erstellung des Primärprogramms in ASSEMBLER-Sprache und die anschließende Übersetzung "von Hand" wird in einem Protokoll niedergelegt. Ein routinierter Anwender benützt für dieses Protokoll diejenige äußere Form, die bei der später zu behandelnden Computer-Assemblierung genormt ist: dieses vom Computer ausgedruckte Übersetzungsprotokoll heißt "Assembler-Listing" (Assemblierungs-Liste).

Das Assembler-Listing hat für unser obiges Beispiel folgendes Aussehen:

			ORG	0020H	;ANFANGSADRESSE
					;(SIEHE SPÄTER)
0020	3A2DA1	SUBR:	LDA	OA12DH	;ERSTE ZAHL IN AKKU
0023	2101OE		LXI	H,OEO1H	; ADRESSE IN H,L
0026	86		ADD	M	; ADDITION
0027	77	ERG:	MOV	M,A	; RESULTAT IN SPEICHER
Adresse (hex)	Inhalt (hex)	Symbol. Adresse	Op.-Code	Operand	Kommentar
übersetztes Programm im Maschinencode		P r i m ä r p r o g r a m m in ASSEMBLER-Sprache			

Das genormte Protokoll ist gegliedert in 6 Spalten:

In Spalte 3 - 6 steht das Primärprogramm in ASSEMBLER-Sprache mit den formalen Regelungen, die für das Assembler-Übersetzungsprogramm eingehalten werden müssen (z.B. nachgestelltes H bei Hexadezimalzahlen usw., siehe nächstes Kapitel). Diese Spalten werden als erste ausgefüllt.

In Spalte 1 und 2 steht das in den Maschinencode übersetzte Programm (Objektprogramm). Spalte 1 enthält die erste Adresse des jeweiligen Befehls, in Spalte 2 steht der Maschinencode des Gesamtbefehls hexadezimal aneinandergereiht (ein 1-Byte-Befehl enthält in dieser Spalte 2 hexadezimale Stellen, ein 3-Byte-Befehl 6 Stellen). Diese beiden Spalten werden also bei der "Übersetzung von Hand" Zeile für Zeile ausgefüllt. Beachte: In der 2. Spalte des Assembler-Listing steht das niedrigere Byte eines Operanden vor dem höheren Byte.

Die symbolischen Adressen in der 3. Spalte sind nichts weiter als Markierungen dieser Zeile und weitgehend dem Programmierer freigestellt. Sie sind dann notwendig, wenn an anderer Stelle des Programms auf diese Zeile Bezug genommen werden soll, ohne daß der Programmierer schon die endgültigen Adressen im Speicher festgelegt hat. Beim Assemblieren (von Hand oder mittels Computer) wird dann jeder symbolischen Adresse eine absolute Adresse zugeteilt.

Die Kommentare in der 6. Spalte sind eine Hilfe für den Programmierer, um das Protokoll übersichtlicher zu gestalten (wichtig!).

Das so von Hand erstellte Protokoll ist dann identisch mit dem vom Computer ausgedruckten "Assembler-Listing".

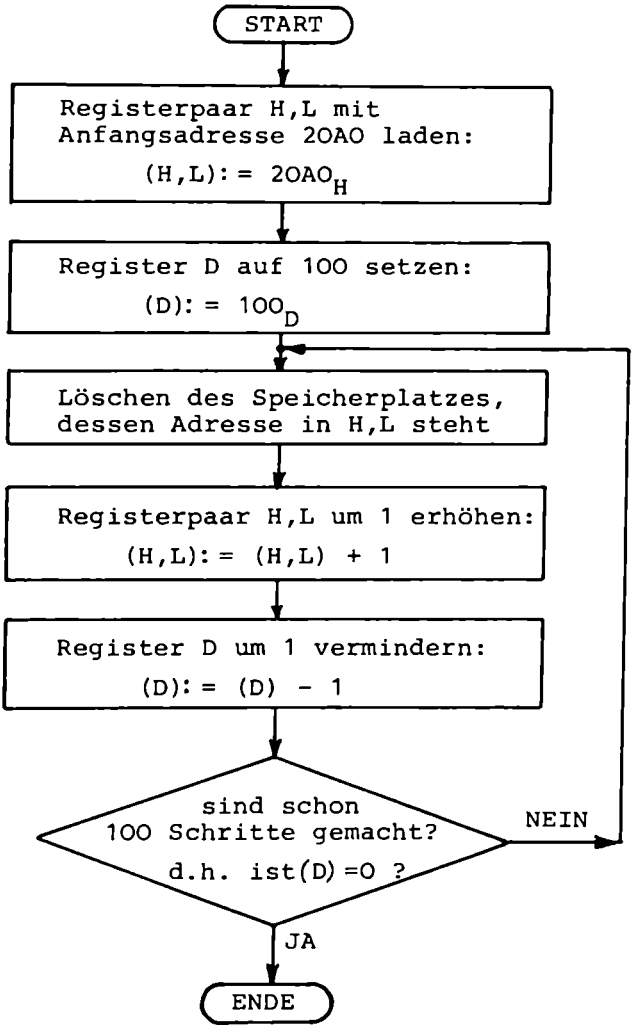
8.12) Zweites Programmbeispiel

Aufgabe: Es soll der Inhalt von 100 aufeinanderfolgenden RAM-Speicherplätzen gelöscht werden. Der erste dieser Plätze habe die hexadezimale Adresse 20A0, es soll also gelöscht werden bis zur hexadezimalen Adresse 2103 .

Die Erstellung des Programms für diese Aufgabe wird wesentlich erleichtert, wenn man sich den Programmablauf zunächst in Form

eines Diagramms darstellt: Programmablaufplan oder Flußdiagramm (Flow Chart).
Die Symbole eines solchen Flußdiagramms sind genormt: rechteckige Kästchen bedeuten "allgemeine Operation", rautenförmige Kästchen bedeuten "Verzweigung" (Entscheidung). Das Flußdiagramm kann Schleifen enthalten.(siehe DIN 66001)
Nach der Erstellung des Flußdiagramms wird dieser Programmablaufplan in ein Programm aus ASSEMBLER-Befehlen übersetzt.

Das Flußdiagramm für die vorliegende Aufgabe sieht folgendermaßen aus:



Das aus diesem Flußdiagramm folgende Primärprogramm in ASSEMBLER-Sprache und das in die Maschinensprache übersetzte Objektprogramm wird in der Protokollform des Assembler-Listing niedergelegt und hat folgende Gestalt:

P R I M Ä R P R O G R A M M

```
                          ORG   00AEH      ; ANFANGSADRESSE
00AE   21A020             LXI   H,20A0H    ; ADRESSE NACH H,L
00B1   1664               MVI   D,100D     ; ANZAHL NACH D
00B3   3600     LOESC:    MVI   M,0        ; LÖSCHE SPEICHER
00B5   23                 INX   H          ; NÄCHSTE ADRESSE
00B6   15                 DCR   D          ; ZÄHLER ERNIEDRIGEN
00B7   C2B300             JNZ   LOESC      ; ZURÜCK SPRINGEN FALLS
                                           ; ZÄHLER NICHT 0
```

Im Registerpaar H,L steht jeweils die Adresse, deren Inhalt gelöscht werden soll. Dies ist ein Musterbeispiel für den Nutzen der indirekten Adressierung.

Das Register D dient als Zählwerk. Da der Befehl DCR das Nullbit beeinflußt, kann man die Lösch-Schleife durch den Befehl JNZ verlassen, wenn der Inhalt des Registers D Null geworden ist.

Der Vorteil einer symbolischen Adresse wird an diesem Beispiel klar, weil man im Primärprogramm die symbolische Rücksprungadresse "LOESC" für die Schleife bereits einführen kann, bevor die endgültige Adresse im Speicher festgelegt ist. Die endgültige Adresse wird dann erst bei der Assemblierung eingesetzt.

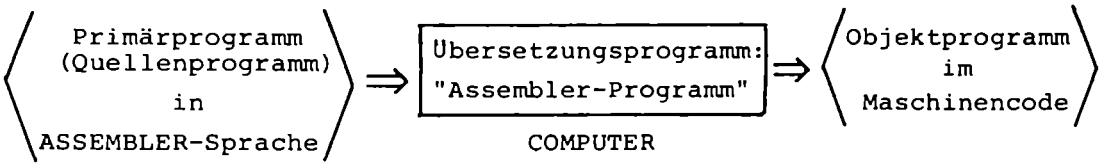
8.2) Assemblierung mittels Computer (Mikrocomputer-Entwicklungssystem)

Für die Übersetzung größerer, in ASSEMBLER-Sprache geschriebener Primärprogramme in die Maschinensprache wird die Verwendung eines übersetzenden Computers (das kann ein allgemeiner Computer oder ein spezielles "Mikrocomputer-Entwicklungssystem" sein) unumgänglich, wenn man den Zeit- und Arbeitsaufwand in Grenzen halten will. Dazu muß der Anwender ein Assembler-Übersetzungsprogramm für den Computer zur Verfügung haben, das auf die spezifische ASSEMBLER-Sprache des betreffenden Mikrocomputer-Typs abgestellt ist. Die Verwendung eines Assemblers setzt voraus, daß das primäre Quellenprogramm bestimmten formalen Konventionen genügt,

die durch das verwendete Assembler-Übersetzungsprogramm vorgegeben sind.

Im folgenden werden wir die formalen Anforderungen an den Satzbau (Syntax) des Primärprogramms besprechen, die einzuhalten sind, damit die maschinelle Übersetzung unter Verwendung des Assembler-Übersetzungsprogramms 8080 der Firmen SIEMENS/INTEL (Assembler 8080) erfolgen kann.

Ein Primärprogramm, das für den Assembler 8080 geeignet sein soll, muß also zum einen die spezifische ASSEMBLER-Sprache des Mikroprozessors 8080 enthalten und zum andern auf die formalen Bedingungen des spezifischen Assembler-Übersetzungsprogramms 8080 abgestellt sein.



8.21) Befehls-Syntax des Primärprogramms

Die formalen Anforderungen an den Satzbau des Primärprogramms, die durch den Assembler 8080 vorgegeben sind, sollen hier kurz zusammengestellt werden. Für ein weitergehendes Studium wird auf die Spezial-Literatur der Firmen SIEMENS und INTEL verwiesen.

Eine Zeile eines Primärprogramms in ASSEMBLER-Sprache, das für den Assembler geeignet ist, hat grundsätzlich folgendes Format, das zusätzlich an einem Beispiel veranschaulicht wird:

Format	Symbolische Adresse: (Marke:)	Operations-Code	Operand	; Kommentar
Beispiel	GET:	MOV	B,A	; BRINGE AKKU ; INS REGISTER B

(Kleinbuchstaben werden vom Assembler als Großbuchstaben interpretiert)

Eine Zeile (ein Befehl) des Primärprogramms hat also 4 (durch Leerzeichen getrennte) unterschiedliche Felder:

Symbolische Adresse (Marke, label): Sie dient zur Markierung eines Befehls im Primärprogramm, bevor die absolute Adresse dieses Befehlscodes schon festgelegt ist. Dem verwendeten Symbol wird dann später bei der Assemblierung die absolute Befehlsadresse zugeordnet.

- kein Pflichtfeld, d.h. es muß nicht notwendigerweise eine symbolische Adresse eingetragen sein
- das Symbol (Marke) muß von einem Doppelpunkt gefolgt sein
- maximale Länge des Symbols: 5 Zeichen (überzählige Zeichen werden ignoriert), das erste Zeichen muß ein Buchstabe sein
- verboten als Symbol sind: Opcode-Bezeichnungen, Registernamen, Pseudobefehle
- korrekte Beispiele: HERE:, GET:, F14F:, LABEL:

Kommentar-Feld: Diese Kommentare sind ein reines Hilfsmittel für den Programmierer. Durch Einfügen solcher erklärender Bemerkungen kann das Primärprogramm übersichtlicher und leichter lesbar gestaltet werden. Der übersetzende Computer ignoriert diese Kommentare, druckt sie jedoch im Assembler-Listing wieder mit aus.

- einzige formale Vorschrift für dieses Feld: Es muß mit einem Strichpunkt (;) beginnen

Operations-Code: In diesem Feld steht die mnemonische Bezeichnung des Operationscodes der Befehle.

Operand: In diesem Feld stehen die Operanden der Befehle, sofern die Befehle Operanden besitzen.
Werden im Operandenfeld Zahlen verwendet, so muß das verwendete Zahlensystem angemerkt werden:

- Hexadezimale Zahlen müssen von dem Buchstaben H gefolgt sein und müssen mit einer Ziffer (0 - 9) beginnen
  z.B.: E301 ⟶ 0E301H

- Dezimalzahlen müssen von dem Buchstaben D oder keinem Buchstaben gefolgt sein
- Dualzahlen müssen von dem Buchstaben B gefolgt sein ("binär")

8.22) Assembler-Anweisungen (Pseudo-Befehle)

In das Primärprogramm (Quellenprogramm) können zusätzliche "Befehle" eingebaut werden, die nur für den Assembler (d.h. für den übersetzenden Computer) als Befehle wirken und nicht als Befehle im Maschinencode des endgültigen Objektprogramms auftauchen. Solche Befehle heißen Pseudo-Befehle oder Assembler-Anweisungen (directives). Wir werden die im Assembler-Programm der Fa. INTEL und SIEMENS vorgesehenen Pseudo-Befehle hier zum Teil vorstellen und verweisen im übrigen auf die Spezial-Literatur dieser Firmen.

a) Die am häufigsten verwendeten Direktiven sind ORG und END:

ORG adr

Dieser Pseudo-Befehl weist den Assembler an, daß er das Objektprogramm bei der Anfangs-Adresse adr beginnen lassen soll. Dem ersten übersetzten Befehl im Maschinencode wird also die Speicheradresse adr zugeordnet (siehe hierzu die früheren Beispiele 8.11) und 8.12)). Enthält ein Primärprogramm keine ORG-Anweisung am Anfang, so beginnt das übersetzte Programm bei der Adresse OOOO H. (ORG = origin)

END

Der END-Befehl zeigt dem Assembler das Ende des Primärprogramms an.

b) Zwei Pseudo-Befehle - EQU und SET - sind vorgesehen, um einem symbolischen Namen (name) einen Wert zuordnen zu können:

name EQU expr
Dem symbolischen Namen "name" wird ein Ausdruck "expr" zugeordnet. Überall, wo später im Primärprogramm der symbolische Name verwendet wird, wird dafür vom Assembler der Ausdruck "expr" eingesetzt. Derselbe Name darf innerhalb eines Programms nur einmal definiert werden. (EQU = equate)

name SET expr
Diese Anweisung ist identisch mit der EQU-Anweisung, nur darf diesmal derselbe Name mehrmals umdefiniert werden.

c) Datendefinition. Mit den Direktiven DB (define byte) und DW (define word) hat man die Möglichkeit, an einer bestimmten Stelle des Speichers konstante Daten abzulegen und zwar 8-Bit-Daten oder — auf 2 Plätzen — 16-Bit-Daten. (Da diese Konstanten zwischen den Befehlen im Speicher liegen können, muß bei der Programmerstellung darauf geachtet werden, daß diese Daten-Speicherplätze nicht versehentlich als Befehle angesprungen werden!):

DB d8
d8 steht für einen Ausdruck mit 8 Bit Länge. Dieser 8-Bit-Ausdruck steht dann im Speicher am Platz mit der laufenden Nummerierung.

DW d16
d16 steht für einen Ausdruck mit 16 Bit Länge. Das niederwertige Byte steht im Speicher am Platz mit der laufenden Nummerierung, das höherwertige Byte am darauffolgenden Platz.

d) Reservierung von Speicherplätzen. Mit der Direktive DS (define storage) wird veranlaßt, daß bei der Übersetzung ein Block von aufeinanderfolgenden Speicherplätzen freigehalten wird und das übersetzte Programm erst am darauffolgenden Speicherplatz fortgesetzt wird:

DS expr
"expr" steht für eine Zahl, welche die Anzahl der zu reservierenden Speicherplätze angibt.

e) Unter den Pseudo-Befehlen ist auch eine bedingte Assemblierung vorgesehen:

```
IF expr
.
.
.   n Befehle
.
ENDIF
```

Der Assembler übersetzt die n Befehle zwischen IF und ENDIF nur dann, wenn der Ausdruck "expr" einen Wert ungleich Null annimmt, sonst läßt er sie aus. (Es kann eine ELSE-Anweisung eingefügt werden).

f) Die Definition eines Makro's in einem Primärprogramm muß zwischen den Pseudobefehlen MACRO und ENDM erfolgen:

```
name MACRO
.
.
.   n Befehle
.
ENDM
```

Die n Befehle zwischen MACRO und ENDM werden vom Assembler als ein Makro-Programm mit dem Namen "name" interpretiert und nach Aufruf an entsprechender Stelle des Objektprogramms eingesetzt.

8.23 Beispiel für ein Assembler-Listing (Assemblierungs-Protokoll)

Um eine Vorstellung zu vermitteln, wie das von einem Computer ausgedruckte Übersetzungs-Protokoll aussieht, wird zum Schluß das Programm für eine elektronische Geschwindigkeitsmessung in der Form des Assembler-Listings wiedergegeben, ohne daß auf den Inhalt des Programms näher eingegangen wird.

Es soll hier lediglich die äußere Form des Assembler-Listings vorgestellt werden und auf die Verwendung von symbolischen Adressen (VORN:, NBH:, etc.), sowie von Pseudo-Befehlen (ORG, EQU, END) im Primärprogramm aufmerksam gemacht werden. Wie man sieht, werden bei der Übersetzung durch den Assembler den symbolischen Adressen absolute Adressen im Objektprogramm zugeordnet. Den "Namen" (LIS, BLITZ, etc.) werden durch die EQU-Anweisungen Zahlen zugeordnet. Eine sinnvolle Ausnützung der Kommentar-Spalte macht das Programm offensichtlich "leserlicher".

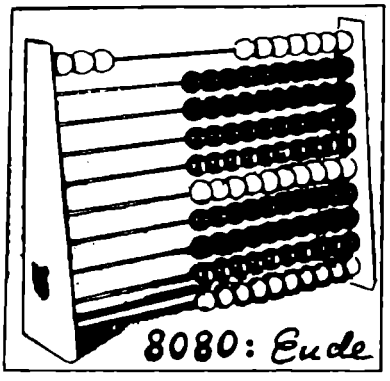
Assembler-Listing:
==========

```
                    ;BEISPIEL: GESCHWINDIGKEITSMESSUNG
                    ;LS-ABSTAND 100 CM
                    ;
0016                LIS     EQU 16H     ;EINGABE KANAL C
0016                BLITZ   EQU 16H     ;AUSGABE KANAL C
0014                ZASCH   EQU 14H     ;EINGABE KANAL A
0015                ANZ     EQU 15H     ;AUSGABE KANAL B
3840                CONST   EQU 14400   ;KONSTANTE
                            ;
                            ORG 4000H   ;ANFANGSADRESSE
4000  3E91                  MVI A,91H   ;STEUERWORT FUER
4002  D317                  OUT 17H     ;  8255 EINGEBEN
4004  97             VORN:  SUB A       ;AKKU=0
4005  D316                  OUT BLITZ   ;BLITZ AUSSCHALTEN
4007  210000                LXI H,0     ;ZAEHLER RUECKSETZEN
400A  DB16           NBH:   IN  LIS     ;LICHTSCHRANKE HOLEN
400C  E603                  ANI 3
400E  FE03                  CPI 3
4010  C20A40                JNZ NBH     ;FZG IN MESSSTRECKE:WARTEN
4013  DB16           LSH:   IN  LIS     ;LS HOLEN
4015  0F                    RRC
4016  DA1340                JC  LSH     ;LINKE LS HELL: WARTEN AUF FZG
4019  110100                LXI D,1
401C  19             RSH:   DAD D       ;ZAEHLER + 1
401D  DA4340                JC  DIEND   ;UEBERLAUF,FZG ZU LANGSAM
4020  0E1D                  MVI C,29    ;ZEITSCHLEIFE
4022  0D             WARTE: DCR C
4023  C22240                JNZ WARTE
4026  DB16                  IN  LIS
4028  E602                  ANI 2
402A  C21C40                JNZ RSH     ;RECHTE LS NOCH HELL: SPRINGE
402D  014038                LXI B,CONST;KONSTANTE AUS WEG U. KM/H
4030  79             DISCH: MOV A,C     ;DIVISION
4031  95                    SUB L
4032  4F                    MOV C,A
4033  78                    MOV A,B
4034  9C                    SBB H
4035  47                    MOV B,A
4036  DA4340                JC  DIEND   ;UNTERLAUF = DIV.ENDE
4039  14                    INR D
403A  B7                    ORA A
403B  7A                    MOV A,D
403C  27                    DAA         ;DEZIMALKORREKTUR
403D  57                    MOV D,A
403E  FE99                  CPI 99H
4040  C23040                JNZ DISCH
4043  7A             DIEND: MOV A,D     ;ERGEBNIS IN AKKU
4044  D315                  OUT ANZ     ;MESSWERT AUF ANZEIGE GEBEN
4046  DB14                  IN  ZASCH   ;SOLLWERT HOLEN
4048  2F                    CMA
4049  BA                    CMP D       ;VERGLEICH MIT MESSWERT
404A  D20440                JNC VORN    ;MESSW. WAR KLEINER, KEIN BLITZ
404D  3E10                  MVI A,10H
404F  D316                  OUT BLITZ   ;BLITZ AUSLOESEN
4051  C30440                JMP VORN    ;NEUE MESSUNG BEGINNEN
                            END
```

Nachwort zum 8080

In dieser einführenden Darstellung der Mikroprozessoren und Mikrocomputer am Beispiel des Mikroprozessors 8080 konnte naturgemäß nicht auf die feineren Schaltungseinzelheiten eingegangen werden, deren Kenntnis beim praktischen Einsatz dieser komplexen elektronischen Bausteine erforderlich wird. Es sei deshalb an dieser Stelle auf die ausführlichen Datenbücher der Herstellerfirmen verwiesen. Auch für die Erstellung komplizierterer Programme wird die Verwendung weiterführender Firmen-Literatur empfohlen, um alle Möglichkeiten, die einem Mikrocomputer innewohnen, ausschöpfen zu können.

Ein "Umsteigen" auf einen anderen Mikroprozessor-Typ stellt nach der genauen Kenntnis eines speziellen Typs keine grundsätzliche Schwierigkeit mehr dar. Doch ist die Gewöhnung und Anhänglichkeit an einen schon bekannten Typ eine nicht zu unterschätzende menschliche Eigenheit, so daß immer eine gewisse Barriere zwischen den verschiedenen Mikroprozessor-Typen mit ihren verschiedenen Sprachen besteht. Eine Lösung dieses Problems könnte für die Zukunft in der Verwendung höherer Programmiersprachen bei der Erstellung von Mikrocomputer-Programmen liegen. Diese höheren Programmiersprachen sind weitgehend unabhängig vom verwendeten Mikrocomputer-Typ. Doch haben die höheren Programmiersprachen das Handicap, daß sie "computerferner" arbeiten als die maschinennahen ASSEMBLER-Sprachen, so daß die mit ihnen erstellten Programme im "Real-Zeit-Verhalten", d.h. im Zeitbedarf des Mikrocomputers zur Bewältigung einer Aufgabe, ungünstiger liegen als die optimal auf den speziellen Mikrocomputer zugeschnittenen ASSEMBLER-Primärprogramme; bei schnellen Steuer- oder Meßvorgängen kann dies von entscheidender Bedeutung sein. Schnellere Mikrocomputer mit größerer Wortlänge (mächtigere Befehle) könnten hier eine Lösung bringen in der Zukunft.



Im zweiten Teil des Buches werden 2 moderne 8-Bit-Mikroprozessoren vorgestellt, deren Aufbau und Funktionsweise unmittelbar aus der Kenntnis des 8080 verständlich werden und die auch historisch aus diesem Mikroprozessor hervorgegangen sind: die Mikroprozessoren 8085 und Z80.
Das Studium der Kapitel über den Mikroprozessor 8085 und der Kapitel über den Mikroprozessor Z80 kann unabhängig voneinander erfolgen.

****
9) DER MIKROPROZESSOR 8085
****

9.1) Einführung

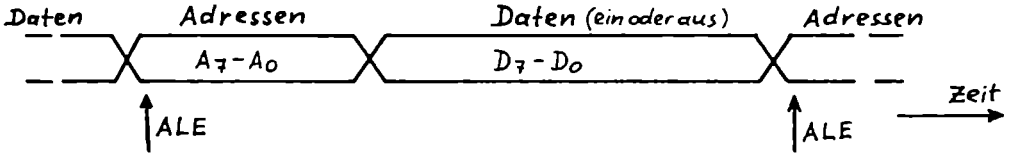
In den folgenden Kapiteln soll ein weiterer 8-Bit-Mikroprozessor vorgestellt werden, der auf dem Prinzip des Mikroprozessors 8080 basiert und aus ihm weiterentwickelt wurde, und der von der Firma INTEL seit 1976 neben dem 8080 auf dem 8-Bit-Markt angeboten wird: der Mikroprozessor 8085. Seine Funktionsweise wird aus der Kenntnis des 8080 unmittelbar verständlich. (Weitere Herstellerfirmen S.23)

Es handelt sich beim Mikroprozessor 8085 um einen noch höher integrierten und noch leistungsfähigeren Prozessor als den 8080. Höher integriert heißt, daß er die gesamte CPU des Systems 8080 - bestehend aus Mikroprozessor 8080, Taktgenerator und Systemsteuerungseinheit - in verbesserter Form auf einem einzigen Chip integriert enthält (vgl. Abb. 12). Darüber hinaus sind beim Mikroprozessor 8085 weitere Funktionen, die beim System 8080 erst durch zusätzliche Bausteine realisiert werden müssen, auf dem Chip integriert: Eine Unterbrechungs-Logik für weitere 4 Unterbrechungs-Eingänge (Interrupts) mit abgestufter Priorität, sowie eine serielle Eingabe/Ausgabe-Schnittstelle.

Bei weiterhin vorhandenen 40 Anschlüssen am Mikroprozessorgehäuse ist der 8085 dann natürlich nicht mehr pin-kompatibel (anschlußkompatibel) mit dem 8080. Doch ist die volle Software-Kompatibilität mit dem 8080 gewährleistet: Der Befehlssatz des Mikroprozessors 8085 umfaßt alle 244 Befehle des 8080, und es kommen noch 2 spezielle Befehle (RIM und SIM) hinzu, welche zum Bedienen der 4 neuen Interrupt-Eingänge und der seriellen Schnittstelle benötigt werden. Die Software-Kompatibilität bedeutet, daß sämtliche für das System 8080 entwickelten Programme (- Software ist in der Erstellung meist sehr zeit- und damit kostenintensiv! -) auf das System 8085 unverändert übernommen werden können und auch die Software-Entwicklungshilfsmittel weiterhin für das neue System verwendbar sind (der 8085 ist "aufwärtskompatibel" zum 8080).

Das Beibehalten von 40 Anschlüssen am Mikroprozessorgehäuse bei höherer Systemintegration und zusätzlich hineingepackten Funktionen ist nur durch Anwendung eines schaltungsmäßigen Tricks möglich: Die 8 Datenleitungen des Mikroprozessors 8085 werden doppelt belegt! Im Zeit-Multiplex-Verfahren werden sowohl die Daten D_0- D_7 als auch die 8 unteren Adressen A_0- A_7 auf dem kombinierten Adress/Datenbus des 8085 abwechselnd übertragen. Der momentane Zustand des Adress/Datenbusses wird über ein Steuersignal (ALE) den übrigen Bausteinen des Mikrocomputersystems angezeigt (ALE = Address Latch Enable) (siehe später).

Multiplex - Adress/Datenbus



Dieser Multiplex-Adress/Datenbus des Mikroprozessors 8085 ist der augenfälligste Unterschied zum 8080! Die 8 oberen Adressen A_8- A_{15} behalten ihren eigenen Adressbus.

Während von der CPU des Systems 8080 4 Steuersignale $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, $\overline{\text{I/OR}}$, $\overline{\text{I/OW}}$ zur Aktivierung der Speicher- und Eingabe/Ausgabe-Bausteine ausgesandt werden, begnügt sich der Mikroprozessor 8085 hierfür mit 3 Steuersignalen $\overline{\text{RD}}$ (Lesen), $\overline{\text{WR}}$ (Schreiben) und IO/$\overline{\text{M}}$ (Unterscheidung zwischen Speicher und Eingabe/Ausgabe-Einheit).

Damit erhalten wir in Abb. 12 die vergleichende Darstellung der Zentraleinheit des Systems 8080 (bestehend aus Mikroprozessor 8080, Taktgenerator 8224 und Systemsteuerungseinheit 8228) und der Zentraleinheit des Systems 8085 (bestehend aus dem Mikroprozessor 8085).

Der Mikroprozessor 8085 wird in 3 verschieden schnellen Versionen angeboten: 8085 AH, 8085 AH-2 und 8085 AH-1. Die Befehlszykluszeit beträgt in der Normalversion 8085 AH: 1,3 µs (3 MHz Taktfrequenz), in der Version 8085 AH-2: 0,8 µs (5 MHz Taktfrequenz); am schnellsten ist die Version 8085 AH-1, ihre Befehlszykluszeit beträgt

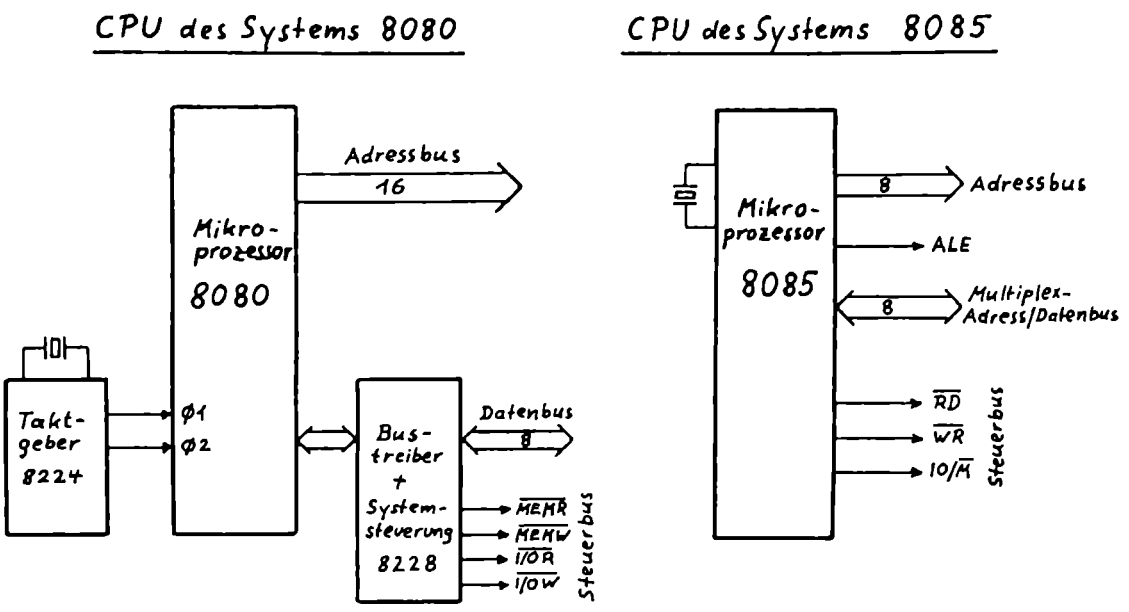


Abb. 12: Gegenüberstellung der Zentraleinheit des Systems 8080 und des Systems 8085
(es sind nur die wichtigsten Leitungen eingezeichnet)

nur noch 0,67 µs bei einer Taktfrequenz von 6 MHz, was einer Taktlänge von 0,17 µs entspricht!

Eine angenehme Verbesserung beim Mikroprozessor 8085 ist, daß er nur noch eine einzige Versorgungsspannung +5 V benötigt (gegenüber 3 Spannungen beim 8080).

Mit der Entwicklung des Systems 8085 wurde die Anzahl der Bausteine, die zum Aufbau der CPU benötigt werden, verringert. Diesem Ziel der Verringerung der Zahl von Systembausteinen diente auch die Entwicklung von neuen kombinierten Speicher-Ein/Ausgabe-Bausteinen für das System 8085:
Der Baustein 8355 ist ein kombinierter ROM-E/A-Baustein, welcher sowohl einen 2K x 8 Bit ROM-Speicher als auch zwei 8 Bit E/A-Kanäle enthält; der Baustein 8755 ist die hierzu pinkompatible EPROM-Version. Der Baustein 8155/8156 ist ein kombinierter RAM-E/A-Zähler-Baustein, welcher sowohl einen 256 x 8 Bit RAM-Speicher als auch drei E/A-Kanäle (2 x 8 Bit und 1 x 6 Bit), sowie einen 14-Bit-Zähler enthält.

Diese neuen Peripheriebausteine können direkt an das im Multiplex-Betrieb arbeitende Bussystem des Mikroprozessors 8085 angeschlossen werden. Dadurch ist es möglich, wie wir sehen werden, mit nur 3 Bausteinen ein Minimalsystem aufzubauen, das alle Eigenschaften eines leistungsfähigen Mikrocomputers enthält, und damit neue Dimensionen der Systemverkleinerung zu erreichen.

Neben der Verwendung dieser speziellen, kombinierten Speicher-E/A-Bausteine ist es jedoch insbesondere beim Aufbau größerer Mikrocomputersysteme zweckmäßig, den Mikroprozessor 8085 mit den uns vom System 8080 her bekannten Standard-Bausteinen (ROM-, RAM-, E/A-Bausteine) zusammenzuschalten, indem man den vollständigen Adressbus aus dem Multiplex-Adress/Datenbus des 8085 durch "Demultiplexen" wiederherstellt (siehe später).

Zusammenfassend seien noch einmal die charakteristischen Daten des 8085 aufgeführt:

- 8-Bit-Mikroprozessor in HMOS-Technologie *) im 40-Pin-Gehäuse
- 3 Versionen: 8085 AH, Zykluszeit 1,3 µs (3 MHz Taktfrequenz)
  *) 8085 AH-2, Zykluszeit 0,8 µs (5 MHz Taktfrequenz)
  8085 AH-1, Zykluszeit 0,67µs (6 MHz Taktfrequenz)
- Software-kompatibel mit dem 8080, 2 neue Befehle RIM und SIM
- Taktgenerator und Systemsteuerungseinheit integriert
- Multiplex-Adress/Datenbus
- +5V Betriebsspannung
- Insgesamt 5 Interrupt-Eingänge mit Prioritätsabstufung
- Serielle Ein/Ausgabe-Schnittstelle integriert
- Anschluß der Standard Speicher- und Ein/Ausgabe-Bausteine des Systems 8080 durch Demultiplexen des Adress/Datenbusses möglich
- 3-Chip-Minimalsystem möglich (mit 8155/8156 und 8355/8755)

---

*) die früheren Versionen 8085 A und 8085 A-2 wurden noch in NMOS-Technologie gefertigt;
daneben werden auch verlustleistungsarme CMOS-Versionen des 8085 angeboten

9.2) Der Aufbau des Mikroprozessors 8085

In Abb. 13 ist das Blockdiagramm des internen Aufbaus des Mikroprozessors 8085 wiedergegeben. Wenn man den Funktionsaufbau des Mikroprozessors 8085 mit dem des Mikroprozessors 8080 (Abb.5, S.49) vergleicht, so fällt die Ähnlichkeit der beiden Funktions-Blockdiagramme unmittelbar ins Auge.

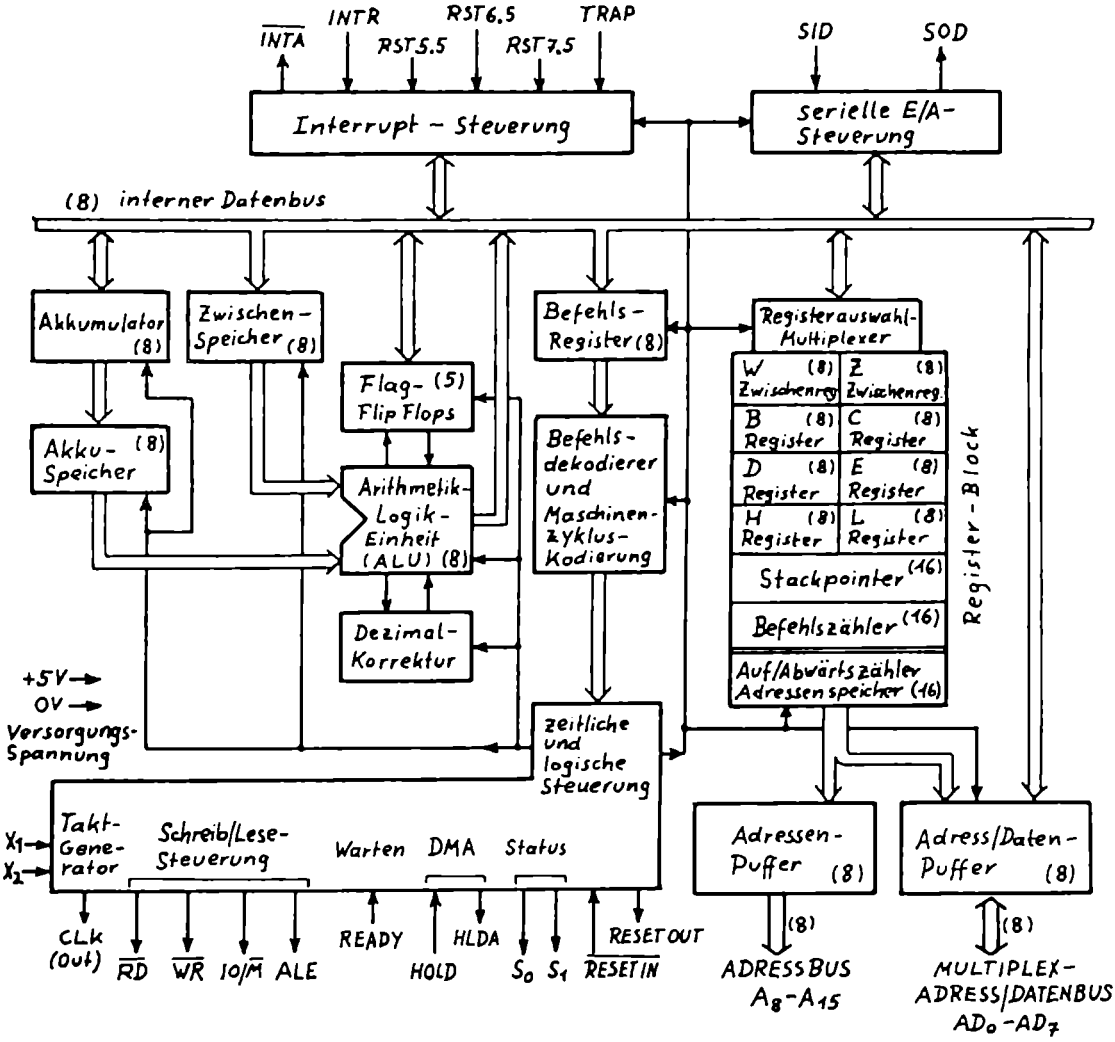


Abb.13: Blockdiagramm des internen Aufbaus des Mikroprozessors 8085

Neu im Blockdiagramm des 8085 ist 1.) die Interrupt-Steuerungseinheit mit jetzt 5 Unterbrechungseingängen, 2.) die serielle Ein/Ausgabe-Steuerung, 3.) die erweiterte Einheit zur zeitlichen und logischen Steuerung, die jetzt auch den Taktgenerator mit enthält (bis auf den Quarzkristall), sowie 4.) der kombinierte Adress/Datenbus-Puffer, von dem der Multiplexbus AD_0- AD_7 ausgeht (die Adressen A_8- A_{15} behalten ihren eigenen Puffer).

Die übrigen Funktionselemente des 8085, die in Abb. 13 auftreten, sind uns vom 8080 her vertraut. So sind z.B. alle universellen Register (A, B C, D E, H L) und alle speziellen Register (Befehlsregister, Befehlszähler, Stapelzeiger, Flag-Register) formal gleich geblieben.

Der im Mikroprozessor 8085 integrierte Generator zur Erzeugung des System-Takts (Clock) wird von einem externen Quarzkristall in seiner Frequenz bestimmt (Anschlüsse X_1 und X_2). Bei geringeren Ansprüchen an die Frequenzkonstanz kann hier auch ein externes frequenzbestimmendes LC- oder RC-Netzwerk angeschlossen werden. Die Oszillatorfrequenz ist beim 8085 doppelt so hoch wie die aus ihr erzeugte Taktfrequenz (Clock). Die maximale Taktfrequenz des 8085 A beträgt 3 MHz (Oszillatorfrequenz 6 MHz) die des schnellsten Typs 8085AH-1 beträgt 6 MHz (Oszillatorfrequenz 12 MHz). Die minimale Taktfrequenz ist bei beiden Typen auf 500 kHz festgelegt (wie beim 8080).

Das Interrupt-System des Mikroprozessors 8085 und die serielle Datenschnittstelle werden in gesonderten Abschnitten behandelt.

Die 40 Anschlußleitungen, über die der Mikroprozessor 8085 mit dem übrigen System in Verbindung steht, sind in Abb. 14 zusammengestellt. Sie haben im einzelnen folgende Funktionen:

- Adressbus mit 8 Leitungen: Er überträgt die höheren 8 Bit der Adresse A_8- A_{15}.
- Multiplex-Adress/Datenbus mit 8 Leitungen AD_0- AD_7: Er überträgt abwechselnd die Daten D_0- D_7 und die niederen 8 Bit der Adresse A_0- A_7. Die Adressen A_0- A_7 erscheinen jeweils im 1. Takt eines Maschinenzyklus auf dem Multiplexbus.

- ALE (Address Latch Enable): Eine Ausgangsleitung, deren Signal den Zeitpunkt anzeigt, zu dem sich Adressen auf dem Multiplexbus befinden (siehe auch Kap.10.1).
- Der eigentliche Steuerbus mit 3 Leitungen $\overline{RD}$, $\overline{WR}$ und IO/$\overline{M}$:
  $\overline{RD}$: Low-aktives Lesesignal für Speicher und E/A.
  $\overline{WR}$: Low-aktives Schreibsignal für Speicher und E/A.
  IO/$\overline{M}$: Unterscheidungssignal zwischen Speicher und E/A-Einheit. Mit IO/$\overline{M}$ = 0 wird der Speicher aktiviert, mit IO/$\overline{M}$ = 1 die E/A-Einheit.
- 5 Interrupt-Anforderungs-Eingänge: In der Prioritätenreihenfolge TRAP, RST 7.5, RST 6.5, RST 5.5, INTR .(INTR entspricht dem Eingang INT des 8080).
- $\overline{INTA}$ (Interrupt-Acknowledge) hat die gleiche Funktion wie beim 8080: Bestätigung, daß der Interrupt INTR (nur dieser!) angenommen wurde.
- HOLD: Wie einst beim 8080 zur DMA-Anforderung (Direct Memory Access = direkter Speicherzugriff).
- HLDA (Hold-Acknowledge): HOLD-Bestätigung.
- READY: Mit READY = 0 kann ein Wartezustand des Prozessors angefordert werden. (Entspricht RDYIN beim 8080).
- $\overline{RESET\ IN}$: Rückstell-Eingang (Low-aktiv).
- RESET OUT: Ein Ausgangssignal, das zum Rückstellen anderer Systembausteine verwendet werden kann.
- X_1, X_2: Anschlüsse für den externen Quarzkristall (oder LC- bzw. RC-Netzwerk).
- CLK: Ausgang, an dem das intern erzeugte Taktsignal abgenommen werden kann.
- SID (Serial Input Data): Serieller Dateneingang.
- SOD (Serial Output Data): Serieller Datenausgang.

- S_1, S_0: 2 Status-Leitungen, die den augenblicklichen Zustand des Mikroprozessors anzeigen:

S_1	S_0	
0	0	HALT
0	1	Schreiben
1	0	Lesen
1	1	Befehls-Opcode Holen (M1)

- +5 V und 0 V: Versorgungsspannung

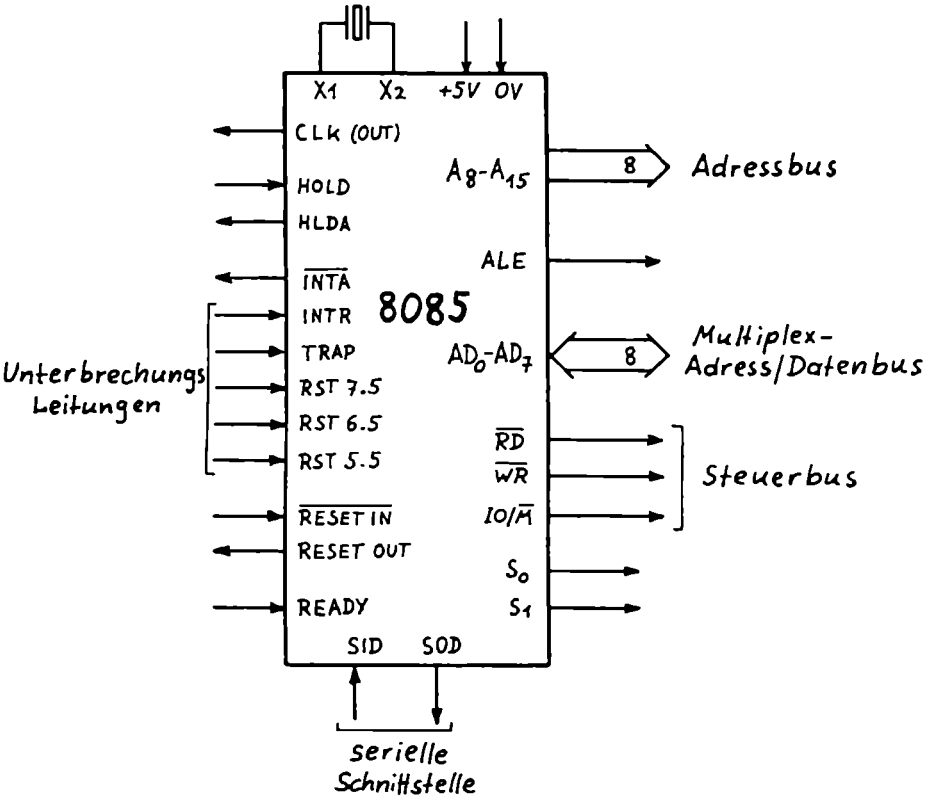


Abb. 14: Die 40 Anschluß-Leitungen des Mikroprozessors 8085

9.3) Das Interrupt-System des Mikroprozessors 8085

Die Unterbrechungs-Möglichkeiten (Interrupts) wurden beim Mikroprozessor 8085 gegenüber dem 8080 wesentlich erweitert. Zu dem Interrupt-Eingang INTR (Interrupt Request), der dem Eingang INT des 8080 entspricht, kommen weitere vier Unterbrechungseingänge hinzu, die untereinander fest vorgegebene Prioritäten besitzen (vgl. Abb. 15). Von den 5 Interrupt-Eingängen besitzt TRAP die höchste Priorität, dann folgen in absteigender Reihenfolge die RESTART-Interrupts RST 7.5, RST 6.5, RST 5.5 und mit niederster Priorität INTR. Liegen mehrere Interrupt-Anforderungen gleichzeitig an, so wird die Anforderung mit der höchsten Priorität angenommen. (Die Priorität bezieht sich nur auf gleichzeitiges Eintreffen von Interrupt-Anforderungen, jede Interrupt-Routine kann durch nachfolgende andere Interrupt-Anforderungen ihrerseits wieder unterbrochen werden, solange dies nicht durch Maskierung ausgeschlossen wird.)

Wird eine Unterbrechung angenommen, so springt der Mikroprozessor in ein Unterprogramm mit einer bestimmten Anfangsadresse (Abb. 15). Die Anfangsadresse wird beim Unterbrechungseingang INTR

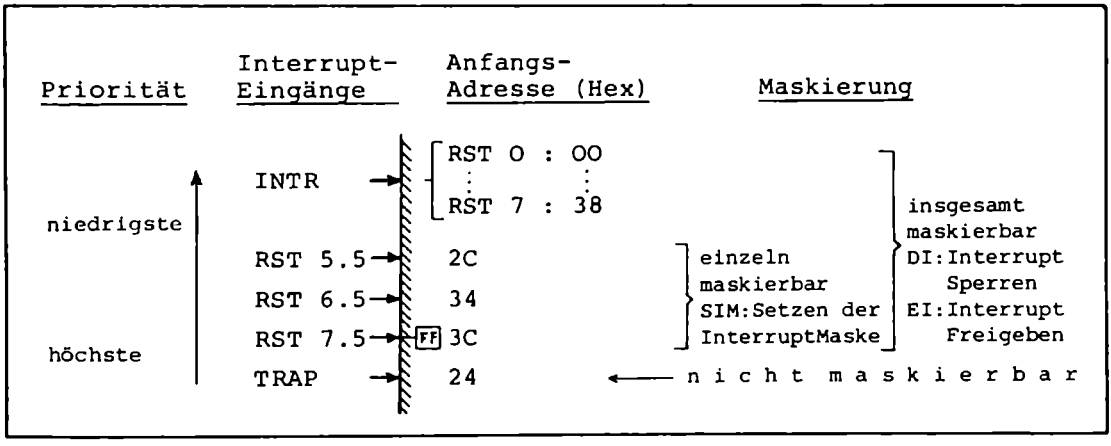


Abb. 15: Das Unterbrechungssystem des Mikroprozessors 8085

wie einst beim 8080 durch das anfordernde periphere Gerät aus 8 Restart-Adressen zwischen OO und 38(H) ausgewählt (Kap. 6.6). Dagegen ist die Anfangsadresse bei den 4 neuen Interrupt-Eingängen intern im Mikroprozessor festgelegt ("Vektorinterrupt"): RST 5.5 hat als Startadresse 2C(H), RST 6.5 → 34(H), RST 7.5 → 3C(H) TRAP → 24(H). Die 4 neuen Restart-Adressen wurden zwischen den alten Restart-Adressen eingeschoben, so daß zwischen zwei Restart-Adressen teilweise nur noch 4 Adressenplätze im Speicher zur Verfügung stehen (z.B. zwischen RST 5.5 und RST 6). Das reicht aber gerade aus, um einen Sprung zur eigentlichen Unterbrechungs-Service-Routine einzuprogrammieren.
Auch die neuen Interrupts retten automatisch den aktuellen Befehlszählerstand in den STACK, bevor die Restart-Adresse angesprungen wird, so daß nach dem Rücksprungbefehl 'RET' das Programm automatisch an der Unterbrechungs-Stelle fortgesetzt wird.

Maskierbare Interrupts

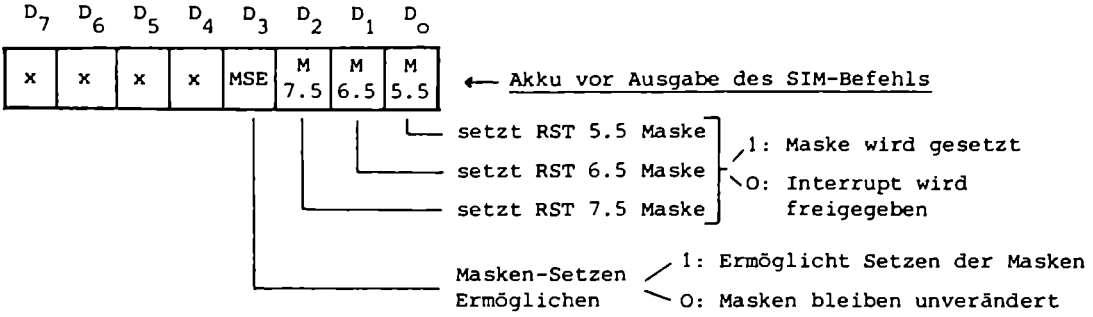
Außer dem TRAP-Interrupt sind alle übrigen Interrupts "maskierbar", d.h. durch Befehle kann die Annahme einer von außen ankommenden Unterbrechungsanforderung verhindert werden (Abb. 15).

Der Unterbrechungseingang TRAP besitzt also nicht nur die höchste Priorität, er ist auch nicht maskierbar (auch nicht durch den allgemeinen Maskierungsbefehl DI). TRAP kann also dazu benützt werden, "Rettungsroutinen" auszulösen, etwa bei Spannungsausfall aktuelle Daten in einen batteriegepufferten RAM-Speicher zu überschreiben, oder bei Prozeßsteuerungen auf schwerwiegende Fehler im Prozeßablauf sofort zu reagieren.

Die übrigen 4 Interrupts RST 5.5, RST 6.5, RST 7.5 und INTR können insgesamt durch den Befehl DI (Disable Interrupt) gesperrt werden. Der Befehl EI (Enable Interrupt) gibt die Unterbrechungsmöglichkeiten insgesamt wieder frei. Bei diesen uns schon bekannten Befehlen wird im Mikroprozessor das Interrupt-Enable-FlipFlop (INTE) gesetzt (durch EI) bzw. rückgesetzt (durch DI). (Vgl. hierzu Kapitel 6.6).

Darüber hinaus können die 3 Restart-Interrupts RST 5.5, RST 6.5 und RST 7.5 auch einzeln maskiert werden. Hierzu besitzt jeder

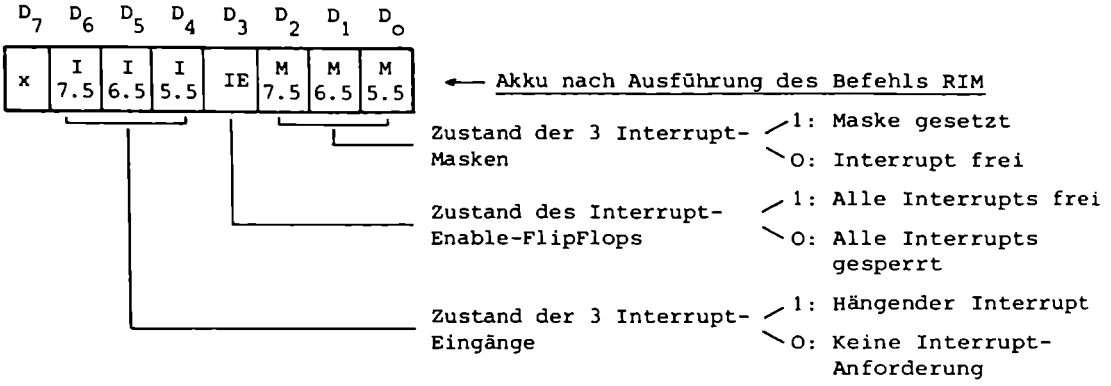
dieser 3 Interrupt-Eingänge eine "Maske" (eine Logikschaltung), die programmiert werden kann. Eine 1 an der Maske "setzt" diese Maske, sperrt also die Interrupt-Möglichkeit, eine O gibt die Interrupt-Leitung frei. Die 3 Masken M 5.5, M 6.5 und M 7.5 werden durch die 3 niedersten Bits des Akkumulators auf den Befehl SIM (Set Interrupt Masks) hin gesetzt oder rückgesetzt. Vor der Ausgabe des Befehls SIM muß also der Akku mit den gewünschten Bits D_0, D_1, D_2 geladen worden sein.



Das Bit D_3 im Akku regelt generell, ob bei der Ausführung des SIM-Befehls die Masken beeinflußt werden können oder nicht:
MSE = Mask Set Enable = Masken-Setzen Ermöglichen.
Bei einer 1 in D_3 werden durch den SIM-Befehl die 3 Masken entsprechend dem Bitmuster D_0, D_1, D_2 gesetzt oder rückgesetzt. Bei einer O in D_3 läßt der SIM-Befehl die 3 Masken unbeeinflußt.

Der Zustand der 3 Interrupt-Masken M 5.5, M 6.5 und M 7.5 kann durch einen Lesebefehl: RIM (Read Interrupt Masks) abgefragt und im Akkumulator in den 3 untersten Bits D_0- D_2 niedergelegt werden. Gleichzeitig wird durch diesen Lesebefehl auch der Zustand des Interrupt-Enable-FlipFlops (das durch EI gesetzt wird) abgefragt und im Bit D_3 des Akkus niedergelegt.
Darüber hinaus wird mit dem Lesebefehl RIM der momentane Zustand der 3 Interrupt-Eingänge RST 5.5, RST 6.5 und RST 7.5 abgefragt und in den Bits D_4, D_5 und D_6 des Akkus niedergelegt. Damit können

anstehende Interrupt-Anforderungen, die z.B. wegen Maskierung noch nicht zum Zuge gekommen sind, erkannt werden: "Hängende" Interrupts (Pending Interrupts).



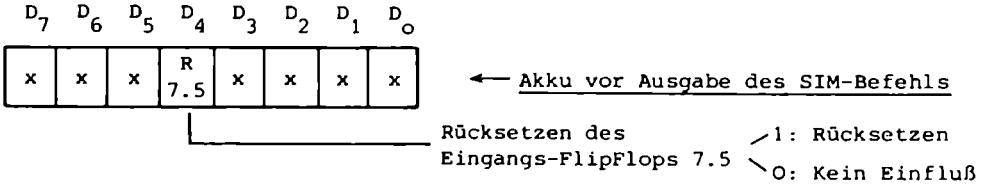
Beachte: Die Annahme irgend eines Interrupts (TRAP, RST 7.5, RST 6.5, RST 5.5, INTR) sperrt alle Interrupteingänge (außer TRAP) so lange, bis ein EI-Befehl gegeben wird!

Spannungspegel für die Interrupt-Anforderungs-Signale

Die Interrupts INTR, RST 5.5, RST 6.5 werden durch High-Pegel am Eingang ausgelöst.

Der TRAP-Interrupt wird durch eine ansteigende Flanke aktiviert und einen nachfolgenden High-Pegel, der mindestens 6 µs andauern muß (solange kann es im ungünstigsten Fall dauern, bis der Mikroprozessor 8085 A die Unterbrechung angenommen hat).

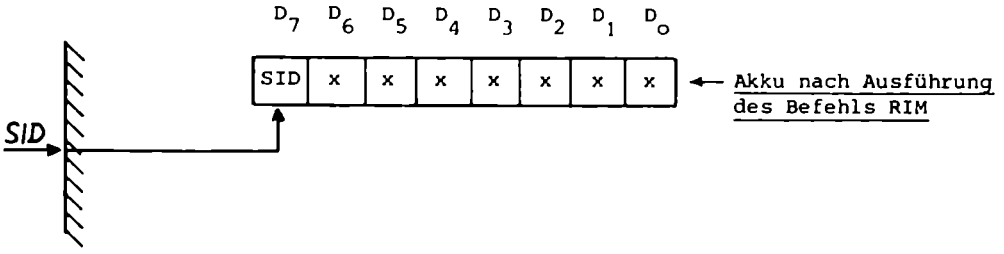
Der RST 7.5 - Interrupt nimmt unter den 5 Interrupt-Eingängen eine Sonderstellung ein. Für seine Anforderung genügt schon eine ansteigende Flanke (flankengesteuerter Eingang), durch welche ein Eingangs-FlipFlop (FF 7.5) gesetzt wird, das die Interrupt-Anforderung so lange speichert, bis sie angenommen wird. Das Rücksetzen dieses Eingangs-FlipFlops 7.5 geschieht entweder automatisch nach Annahme der Interrupt-Anforderung, oder aber das FlipFlop kann im Zuge des SIM-Befehls rückgesetzt werden durch eine 1 im Bit D_4 des Akkumulators:



9.4) Die seriellen Eingabe/Ausgabe-Leitungen SID und SOD

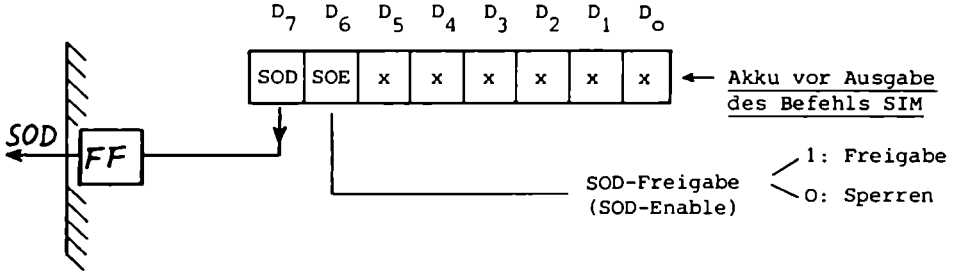
Der Mikroprozessor 8085 enthält integriert eine serielle Datenschnittstelle. Mit den beiden Anschlüssen SID (Serial Input Data) und SOD (Serial Output Data) wird eine einfache Datenübertragung zwischen Peripherie und dem Akkumulator des Mikroprozessors ermöglicht.

Jedesmal, wenn der Befehl RIM erteilt wird, wird der Zustand des SID-Eingangs in das Bit D_7 des Akkumulators eingelesen.



Der Befehl RIM ist also ein "Mehrzweckbefehl", da er auch noch die im vorigen Kapitel beschriebenen Interrupt-Lesefunktionen hat.

Umgekehrt wird mit dem Befehl SIM das Bit 7 des Akkumulators in ein Ausgangs-FlipFlop übertragen, so daß es am SOD-Ausgang zur Verfügung steht.



Durch das Bit D_6 des Akkumulators wird beim SIM-Befehl generell der serielle Datenausgang SOD freigegeben (D_6= 1) oder gesperrt (D_6= 0): SOE = SOD- Enable.
Auch der SIM-Befehl ist ein Mehrzweckbefehl. Man muß also darauf achten, daß nicht unbeabsichtigt Interrupt-Masken geändert werden (Bits D_0- D_4 des Akkus), wenn man den SIM-Befehl zur seriellen Datenausgabe benützt!

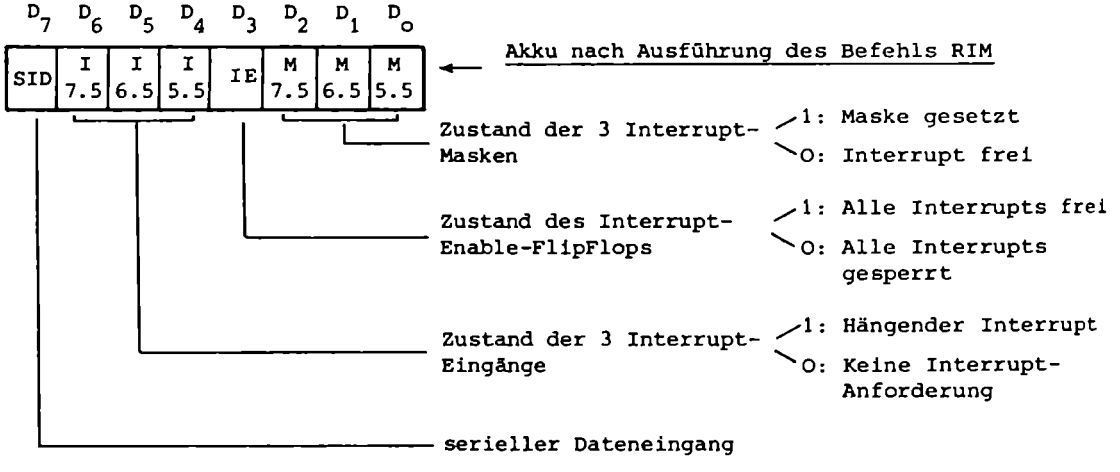
9.5) Der Befehlssatz des Mikroprozessors 8085

Alle Befehle des Mikroprozessors 8080 wurden beim Mikroprozessor 8085 unverändert übernommen. Lediglich der interne zeitliche Ablauf mancher Befehle hat sich gegenüber dem 8080 geändert: Manche Befehle brauchen zur Ausführung beim 8085 eine andere Anzahl von Takten als beim 8080. (Diese abweichende Taktzahl des 8085 ist in der Befehls-Auflistung im Kapitel 7 in Klammern angefügt).

Zwei neue Befehle kommen beim Mikroprozessor 8085 hinzu:
RIM (Operationscode 20(H)) und SIM (Operationscode 30(H)).
Ihre Funktion wurde im einzelnen in den Abschnitten 9.3 und 9.4 besprochen. Hier seien sie noch einmal zusammenfassend dargestellt:

RIM (read interrupt masks)

Ein Mehrzweckbefehl, der Daten in den Akkumulator einliest mit folgender Bedeutung:



Der Opcode des Befehls lautet:

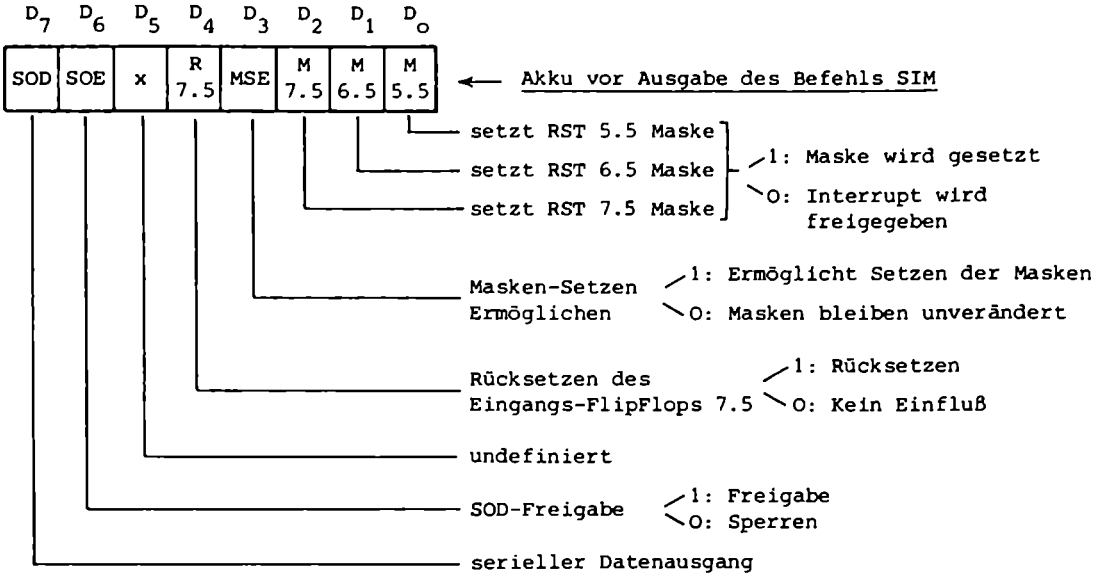
0	0	1	0	0	0	0	0

(20)

- 1 Zyklus, 4 Takte
- veränderte Flags: keine

SIM (set interrupt masks)

Ein Mehrzweckbefehl, der den Bits des Akkumulators (die zuvor eingegeben sein mußten) folgende Funktionen zuweist:



Der Opcode des Befehls lautet:

0 0 1 1 0 0 0 0	(30)

- 1 Zyklus, 4 Takte
- veränderte Flags: keine

10) WIEDERHERSTELLUNG DES VOLLSTÄNDIGEN ADRESSBUSSES UND DER EHEMALIGEN 4 STEUERSIGNALE DES SYSTEMS 8080 AUS DEN SIGNALEN DES MIKROPROZESSORS 8085

Wenn man in der Lage wäre, aus den Ausgangsleitungen des Mikroprozessors 8085 die ehemaligen Bus- und Steuerleitungen des Systems 8080 wiederzugewinnen, hätte man den unmittelbaren Anschluß des Mikroprozessors 8085 an die Schaltungen des Systems 8080 erhalten und könnte damit auf die bekannten Speicher- und Eingabe/Ausgabe-Bausteine und ihre Schaltungen zurückgreifen.

In der Tat ist, wie wir sehen werden,
- die Wiedergewinnung des vollständigen Adressbusses A_0- A_{15} aus dem Multiplex-Adress/Datenbus des 8085, sowie
- die Erzeugung der 4 Steuersignale $\overline{MEMR}$, $\overline{MEMW}$, $\overline{I/OR}$, $\overline{I/OW}$ des Systems 8080 aus den Steuersignalen $\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$ des 8085

mit relativ einfachen Logikschaltungen leicht möglich.

Man kann damit die beim System 8080 gewonnenen Ergebnisse und Schaltungen direkt auf den Mikroprozessor 8085 übernehmen!

10.1) Gegenüberstellung des zeitlichen Ablaufs der Signale auf dem Datenbus und Adressbus beim Mikroprozessor 8080 und 8085

Die Unterschiede im zeitlichen Ablauf der Signale zwischen den beiden Mikrocomputer-Typen 8080 und 8085 sind aus Abb. 16 ersichtlich, in welcher der Signalablauf auf verschiedenen Leitungen des Mikroprozessors 8080 (Abb. 16a) sowie des Mikroprozessors 8085 (Abb. 16 b) dargestellt ist. Als Beispiel ist die Einholung und Ausführung des Befehls "OUT nr" gewählt: Ausgabe von Daten über einen E/A-Kanal mit einer 8-Bit-Kanalnummer "nr".

Zunächst soll der grundsätzliche zeitliche Ablauf des Befehls "OUT nr" erklärt werden:
Im 1. Maschinenzyklus M_1 wird der Opcode des Befehls aus dem Programmspeicher eingeholt, die hierzu notwendige Speicheradresse

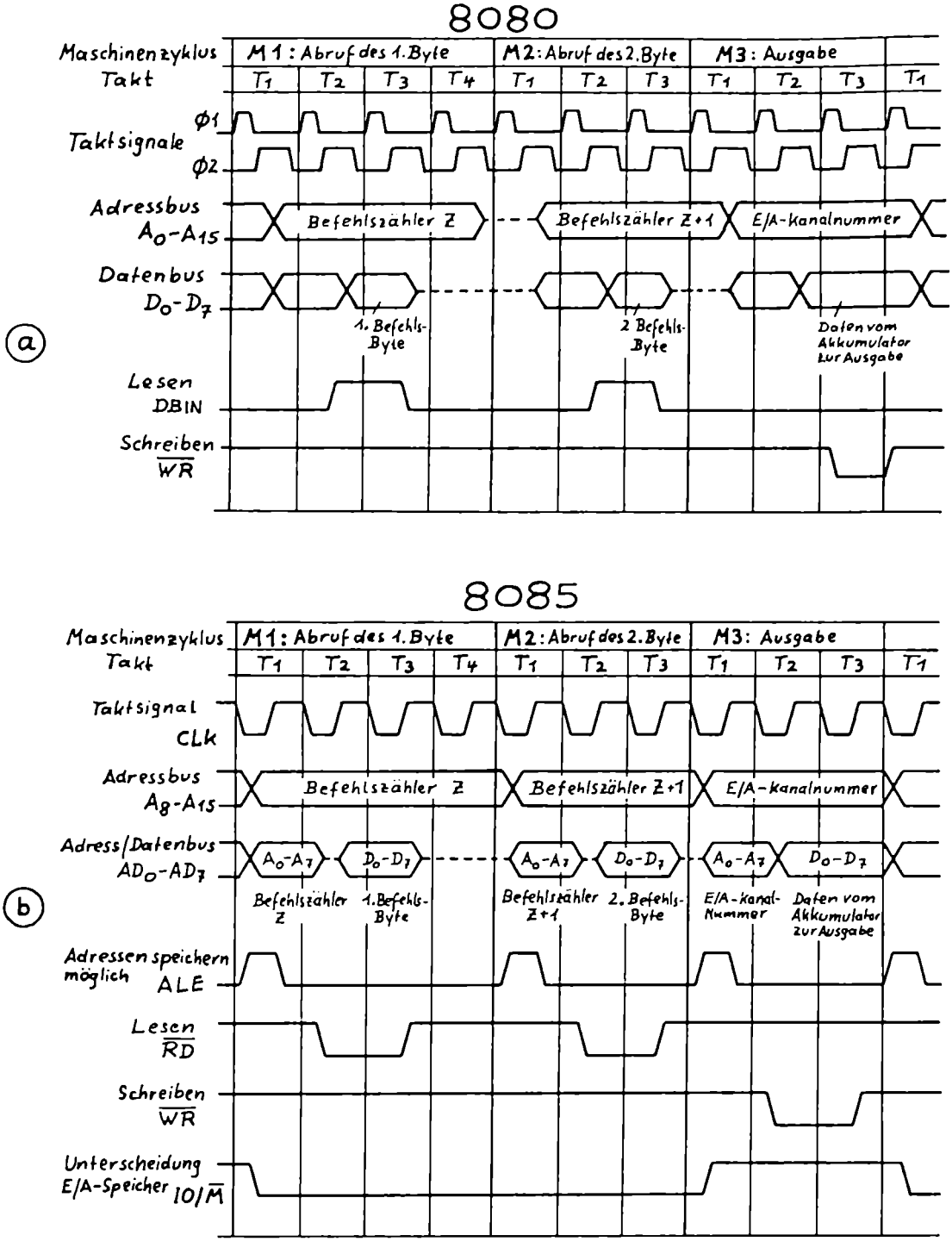


Abb. 16: Zeitlicher Ablauf der Signale beim Ausgabebefehl "OUT nr"

a) beim Mikroprozessor 8080
b) beim Mikroprozessor 8085

wird durch den Befehlszähler angegeben. Der Opcode (Byte 1 des Befehls) wird auf den Datenbus gegeben und von der CPU übernommen.
Im 2. Maschinenzyklus M_2 wird auf gleiche Weise das 2. Byte des Befehls (die 8-Bit-Kanalnummer) auf den Datenbus gegeben, die hierzu vom Befehlszähler ausgegebene Adresse ist um 1 höher.

Im 3. Maschinenzyklus M_3 gibt die CPU die Kanalnummer auf die Adressleitungen, das im Akku enthaltene Datenwort wird anschließend über die Datenleitung an diesen adressierten E/A-Kanal ausgegeben.

a) Zeitlicher Signalablauf des Ausgabe-Befehlszyklus beim 8080.
Beim Mikroprozessor 8080 werden Adressen und Daten voneinander getrennt auf 2 verschiedenen Bussen (16-Bit-Adressbus, 8-Bit-Datenbus) geführt (Abb. 16 a). Die anstehenden Daten werden vom 8080 aus dem Datenbus zum Zeitpunkt DBIN = 1 in die CPU hereingenommen (Lesen). Zum Zeitpunkt $\overline{WR}$ = 0 werden aus der Zentraleinheit Daten über den Datenbus ausgegeben (Schreiben). Zu beiden Zeitpunkten ist auf dem Adressbus die vollständige Adresse A_0 - A_{15} vorhanden.

b) Zeitlicher Signalablauf des Ausgabe-Befehlszyklus beim 8085.
Da beim Mikroprozessor 8085 sowohl Daten als auch die unteren Adressen im Zeit-Multiplex-Verfahren auf dem gemeinsamen Adress/Datenbus übertragen werden, ist nur zu bestimmten Zeitpunkten die vollständige 16-Bit-Adresse auf den Leitungen AD_0- AD_7 / A_8- A_{15} vorhanden (Abb. 16 b). Diese Zeitpunkte (der 1. Takt im jeweiligen Maschinenzyklus) werden durch ein 1-Signal auf der Leitung ALE (Address Latch Enable) angezeigt. Wenn sich Daten auf dem Adress/Datenbus befinden, so wird dies durch ein 0-Signal auf der $\overline{RD}$ (Lesen) bzw. $\overline{WR}$ (Schreiben) Leitung angezeigt. Somit kann immer zum richtigen Zeitpunkt der kombinierte Adress/Datenbus abgefragt werden. Die Entscheidung, ob der Speicher oder die Eingabe/Ausgabe-Einheit aktiviert werden soll, wird durch das Signal IO/$\overline{M}$ getroffen.

Der Ausgabebefehl "OUT nr" enthält als Kanalnummer "nr" eine 8-Bit-Zahl (im Gegensatz zur 16-Bit-Adresse bei Speichern). Beachte, daß die 8-Bit-Adresse eines Kanals beim 8085 sowohl in der oberen (A_8- A_{15}) als auch in der unteren Hälfte (AD_o- AD_7) gleichlautend, also doppelt auftaucht (Zeitpunkt M_3 - T_1). Auch beim 8080 sind die 8-Bit-Kanalnummern auf dem Adressbus doppelt ausgebracht (A_o- A_7 identisch mit A_8- A_{15}). Dies ermöglicht, wie wir später sehen werden, die Übernahme sämtlicher Standard-Eingabe/Ausgabe-Bausteine aus dem System 8080 ohne zusätzlichen Schaltungsaufwand (ohne Demultiplexen!) auf das System 8085.

10.2) Erzeugung des vollständigen Adressbusses A_o-A_{15}

Der vollständige Adressbus A_o-A_{15}, wie er im System 8080 Verwendung findet, kann aus den Ausgangsleitungen des Mikroprozessors 8085 wie folgt aufgebaut werden (Abb.17):
Die oberen 8 Adressen A_8- A_{15} stehen unmittelbar am Ausgang des Mikroprozessors 8085 zur Verfügung. Die unteren 8 Adressen A_o-A_7 werden aus dem Multiplex-Adress/Datenbus AD_o- AD_7 durch "Demultiplexen" gewonnen unter Verwendung des Zwischenspeichers 8212 ("Adressen-Latch").

Wie wir in Abb. 16b gesehen haben, stehen die Adressen A_o- A_7 auf dem Adress/Datenbus zum Zeitpunkt ALE = 1 bereit. Sie werden zu diesem Zeitpunkt (auf Veranlassung des ALE-Signals) in den Zwischenspeicher 8212, der aus 8 Flip-Flops besteht, übernommen und stehen an dessen Ausgang andauernd bis zum Eintreffen des nächsten ALE-Signals zur Verfügung. Der vollständige Adressbus A_o- A_{15} wird also zusammengesetzt aus:

8 Leitungen des Mikroprozessors 8085: A_8- A_{15}
8 Ausgangsleitungen des Adressen-Latch 8212: A_o- A_7 .

Der Adressen-Latch-Baustein 8212 ist ein Standard-Baustein des Systems 8080/85, die detaillierte Zusammenschaltung mit dem Mikroprozessor 8085 ist aus Abb. 17 ersichtlich. Der Anschluß DS2 ist ein High-aktiver Chip-Select-Eingang, der Baustein 8212 wird also durch ALE = 1 aktiviert.

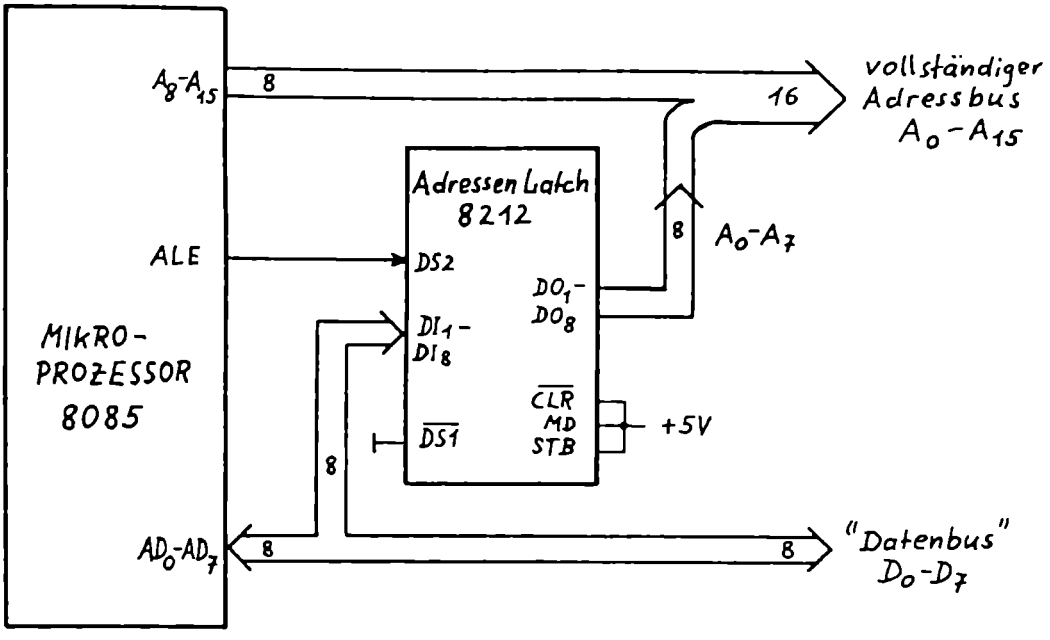


Abb. 17: Schaltung zur Erzeugung des vollständigen Adressbusses A_0- A_{15}

Der verbleibende "Datenbus" D_0- D_7 ist natürlich weiterhin der Multiplex-Bus AD_0- AD_7, auf dem auch noch abwechselnd Adressen stehen. Dies ist jedoch bei der Übermittlung von Daten nicht störend, da der Bus nur dann abgefragt wird, wenn das $\overline{RD}$- bzw. $\overline{WR}$-Steuersignal signalisiert, daß jetzt Daten D_0- D_7 auf dem Bus anstehen (vgl. Abb. 16b).

10.3) Erzeugung der Steuersignale des 8080 ($\overline{MEMR}$,$\overline{MEMW}$,$\overline{I/OR}$,$\overline{I/OW}$) aus den Steuersignalen des 8085

Während der Mikroprozessor 8080 mit 4 Schreib-Lese-Steuersignalen arbeitet ($\overline{MEMR}$, $\overline{MEMW}$, $\overline{I/OR}$, $\overline{I/OW}$) kommt der Mikroprozessor 8085 mit 3 solchen Signalen aus ($\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$).
Beim Mikroprozessor 8085 wird durch eine einzige Steuerleitung IO/$\overline{M}$ zwischen Speicherbausteinen und Eingabe/Ausgabe-Bausteinen unterschieden (vgl. Abb. 16b):

$IO/\overline{M}$ = 1 bedeutet: die Eingabe/Ausgabe-Einheit wird aktiviert,
$IO/\overline{M}$ = 0 bedeutet: Aktivierung des Speichers.

Um aus den 3 Steuersignalen des 8085 die 4 Steuersignale des 8080 wiederzugewinnen, ist eine einfache Logikschaltung entsprechend Abb. 18a) notwendig. Die zugehörige Wahrheitstafel erklärt den logischen Zusammenhang. (Z.B. ist die $\overline{MEMR}$-Leitung durch eine einfache ODER-Verknüpfung der $\overline{RD}$ und $IO/\overline{M}$-Leitung entstanden, d.h. $\overline{MEMR}$ ist nur dann Null, wenn $\overline{RD}$ und $IO/\overline{M}$ Null sind; in diesem Fall wird also der Speicher gelesen).

Diese erforderliche Logik-Verknüpfung kann auch durch den Baustein 8205 realisiert werden, wie Abb. 18b zeigt. Der Baustein 8205 ist der uns schon bekannte "1 aus 8 Dekoder". Die zugehörige Wahrheitstafel zeigt den logischen Zusammenhang zwischen den 3 Eingangsleitungen A_0 - A_2 und den 8 Ausgangsleitungen O_0 - O_7 dieses Bausteins (z.B. ist der Ausgang O_1 nur dann Null, wenn $\overline{RD}$ (A_1) und $IO/\overline{M}$ (A_2) Null sind; dies ist die logische Bedingung für $\overline{MEMR}$).

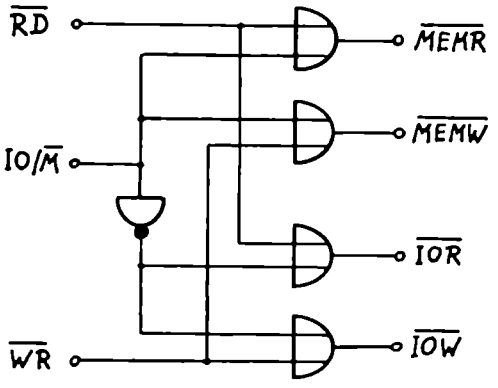
Es ist schon an dieser Stelle zu vermerken, daß meist unmittelbar die 3 Steuersignale des Mikroprozessors 8085 verwendet werden, um die Standard-Bausteine des Systems anzusteuern. Man muß also nicht unbedingt den grundsätzlich möglichen Weg über die Erzeugung der 4 ehemaligen Steuersignale gehen, wenn man die vom System 8080 her bekannten Standard-Bausteine auf das System 8085 übernehmen will.

Wahrheitstabelle

$\overline{RD}$	$\overline{WR}$	IO/$\overline{M}$	$\overline{MEMR}$	$\overline{IOR}$
0	x	0	0	1
0	x	1	1	0
1	x	0	1	1
1	x	1	1	1

$\overline{RD}$	$\overline{WR}$	IO/$\overline{M}$	$\overline{MEMW}$	$\overline{IOW}$
x	0	0	0	1
x	0	1	1	0
x	1	0	1	1
x	1	1	1	1

ⓐ



Schaltalgebraische Gleichungen: $\overline{MEMR} = \overline{RD} + IO/\overline{M}$ | $\overline{IOR} = \overline{RD} + \overline{IO/\overline{M}}$

$\overline{MEMW} = \overline{WR} + IO/\overline{M}$ | $\overline{IOW} = \overline{WR} + \overline{IO/\overline{M}}$

Wahrheitstabelle

Eingänge A			Ausgänge O							
A_0	A_1	A_2	0	1	2	3	4	5	6	7
0	0	0	0	1	1	1	1	1	1	1
1	0	0	1	0	1	1	1	1	1	1
0	1	0	1	1	0	1	1	1	1	1
1	1	0	1	1	1	0	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1
1	0	1	1	1	1	1	1	0	1	1
0	1	1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	1	1	1	0

ⓑ

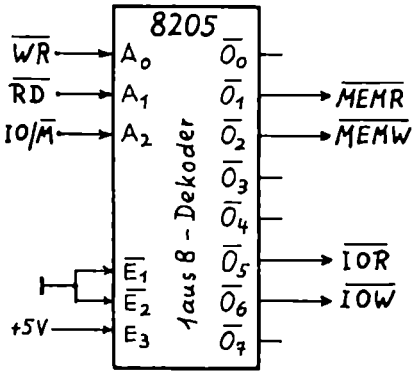


Abb. 18: Erzeugung der Steuersignale des 8080 ($\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$, $\overline{IOW}$) aus den Steuersignalen des 8085 ($\overline{WR}$, $\overline{RD}$, IO/$\overline{M}$)
a) durch ein einfaches Logik-Netzwerk
b) mit Hilfe des "1 aus 8" Dekoder-Bausteins 8205

11) AUFBAU VON MIKROCOMPUTERN MIT DEM MIKROPROZESSOR 8085

Der Aufbau von Mikrocomputern unter Verwendung des Mikroprozessors 8085 kann auf 2 Wegen erfolgen.

Zum einen können die bekannten Standard-Bausteine zum Aufbau eines Mikrocomputersystems 8085 herangezogen werden. Diesen Weg wird man immer dann gehen, wenn größere Computersysteme mit dem 8085 aufgebaut werden sollen und man dabei auf die umfangreiche Palette der Speicherbausteine und Eingabe/Ausgabe-Bausteine sowie Interfacebausteine des Systems 8080/85 zurückgreifen will. Hierzu ist dann die beschriebene Schaltung zur Wiederherstellung des vollständigen Adressbusses erforderlich (insbesondere für die Verwendung von Standard-Speicherbausteinen).

Begnügt man sich jedoch mit der Minimalkonfiguration eines Mikrocomputers 8085 (die für viele Zwecke völlig ausreicht!), so verwendet man vorteilhaft spezielle, für das System 8085 entwickelte kombinierte Speicher-Eingabe/Ausgabe-Bausteine, bei denen der Adress/Datenbus nicht gesondert demultiplext werden muß, und mit denen dann ein erstaunlich einfacher Aufbau eines Mikrocomputers möglich wird. Diese speziellen, hochintegrierten Bausteine 8355/8755 bzw. 8155/8156 werden wir später kennenlernen.

11.1) Mikrocomputersysteme 8085 unter Verwendung von Standard-Bausteinen des Systems 8080/85

Benützt man die Schaltung zur Wiedergewinnung des vollständigen Adressbusses A_0- A_{15}, so kann man auf die bekannten Standard-Bausteine des Systems 8080/85 zurückgreifen, und es können dann die früher beschriebenen Mikrocomputerschaltungen in ähnlicher Weise aufgebaut werden, wobei lediglich der Mikroprozessor 8080 gegen den leistungsfähigeren und höher integrierten Typ 8085 ausgetauscht wird. Verwendet man zudem noch die Schaltungen zur Erzeugung der früheren 4 Steuersignale $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, $\overline{\text{I/OR}}$, $\overline{\text{I/OW}}$, so können die beim System 8080 angegebenen Schaltungen sogar ohne Veränderungen übernommen werden.

a) Zentraleinheit 8085 mit vollständigem Adressbus A_0- A_{15} und mit 4 Steuerleitungen $\overline{MEMR}$, $\overline{MEMW}$, $\overline{I/OR}$, $\overline{I/OW}$

Mit der nachfolgenden Schaltung werden konsequent die Schaltungen des letzten Kapitels übernommen und damit schaltungsmäßig (und didaktisch) der Mikroprozessor 8085 an die Ergebnisse des Mikroprozessors 8080 angeschlossen.

Es wird sowohl der Adress/Datenbus des 8085 demultiplext und damit der vollständige Adressbus A_0- A_{15} wieder hergestellt, als auch die 4 früheren Steuerleitungen $\overline{MEMR}$, $\overline{MEMW}$, $\overline{I/OR}$, $\overline{I/OW}$ aus den 3 Steuerleitungen des 8085 wiedergewonnen. Der Multiplex-Bus wird als Datenbus D_0- D_7 weiterverwendet, denn es ist gewährleistet, daß sich während der aktiven Phase der 4 Steuersignale $\overline{MEMR}$, $\overline{MEMW}$, $\overline{I/OR}$, $\overline{I/OW}$ auf dem Bus Daten und keine Adressen befinden; d.h. der Multiplex-Bus entspricht zu diesen Zeitpunkten dem Datenbus D_0- D_7 des Systems 8080.

Das Demultiplexen des Adress/Datenbusses erfolgt, wie beschrieben, mit dem Adressen-Latch 8212, die Wiedergewinnung der 4 Steuersignale mit dem Dekoder 8205.

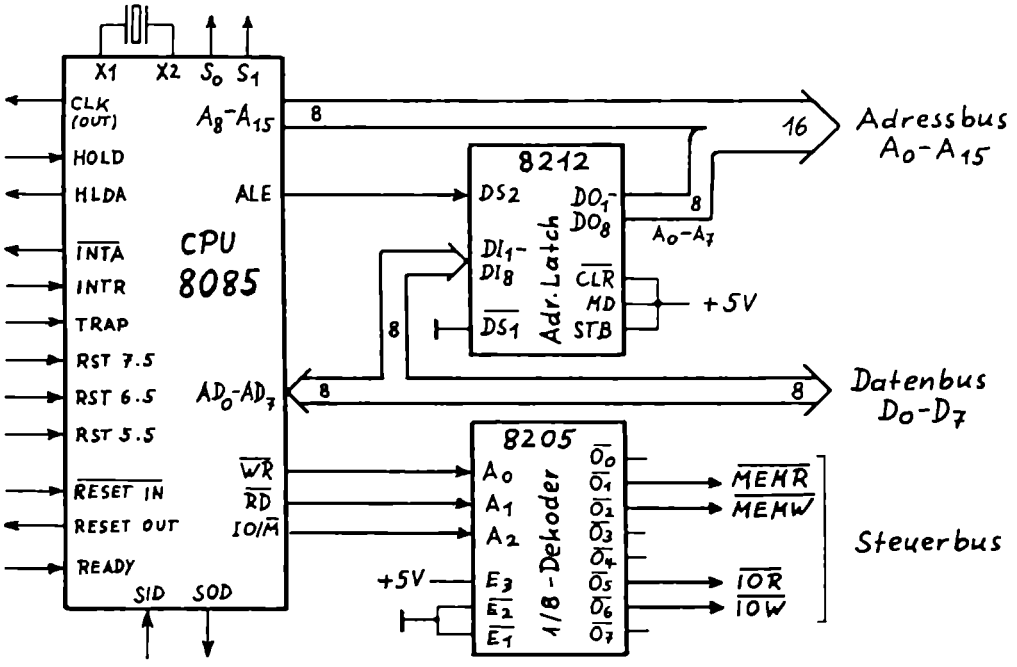


Abb. 19: Zentraleinheit mit dem Mikroprozessor 8085 und vollständigem Adressbus Ao - A15 sowie 4 Steuerleitungen $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$, $\overline{IOW}$

Damit steht mit der in Abb. 19 dargestellten Schaltung eine Zentraleinheit (CPU) zur Verfügung, die gegen die im Blockschaltbild von Kapitel 5 verwendete Zentraleinheit (bestehend aus 8080, 8224 und 8228) voll austauschbar ist. Es können also die im Kapitel 5 beschriebenen Computerschaltungen unverändert übernommen und mit dieser neuen CPU aufgebaut werden, und natürlich können sämtliche aus dem System 8080 bekannten Standard-Bausteine an diese neue CPU ohne schaltungsmäßige Änderungen angeschlossen werden[+].

b) Schaltungen mit dem 8085 unter direkter Verwendung der 3 Steuerleitungen $\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$.

1.) Es ist wie gesagt nicht erforderlich, daß man auf die ehemaligen 4 Steuersignale zurückgeht, wenn man Standard-Bausteine des Systems 8080/85 verwenden will. Vielmehr kann man unmittelbar mit den 3 Steuersignalen $\overline{RD}$, $\overline{WR}$, IO/$\overline{M}$ des 8085 diese Standard-Bausteine logisch ansteuern. Man muß sich dazu klar machen, daß beim System 8085: $\overline{RD}$ das allgemeine Lesesignal, $\overline{WR}$ das allgemeine Schreibsignal und IO/$\overline{M}$ das Auswahlsignal zwischen Speicher- und E/A-Bausteinen bedeutet.

Die Unterscheidung zwischen Speicher und Ein/Ausgabe-Einheit erfolgt dann dadurch, daß man mit dem IO/$\overline{M}$-Signal die Baustein-Freigabe-Eingänge (CE bzw. CS) der einzelnen Standard Speicher- und E/A-Bausteine unmittelbar ansteuert (siehe Abb. 20). Die logische Verknüpfung zwischen der IO/$\overline{M}$-Leitung und den Freigabe-Eingängen muß dabei so ausgelegt werden, daß IO/$\overline{M}$ = 1 die E/A-Bausteine aktiviert und die Speicherbausteine sperrt, während das Signal IO/$\overline{M}$ = 0 die Speicher-Bausteine aktiviert und die E/A-Bausteine unbeeinflußt läßt. (In der Schaltung Abb. 20 ist daher IO/$\overline{M}$ mit dem Low-aktiven $\overline{CE}$-Eingang des Speichers und dem High-aktiven CE-Eingang des E/A-Bausteins verbunden). Mit den Signalen $\overline{RD}$ und $\overline{WR}$ wird der Datenaustausch veranlaßt.

+) Es ist zu beachten, daß bei voller Geschwindigkeit des 8085 AH-1 die Normalversion der Speicher- und E/A-Bausteine zu langsam sein kann. Deshalb gibt es von diesen Bausteinen schnellere Versionen, die an die schnellen Mikroprozessoren direkt angeschlossen werden können.

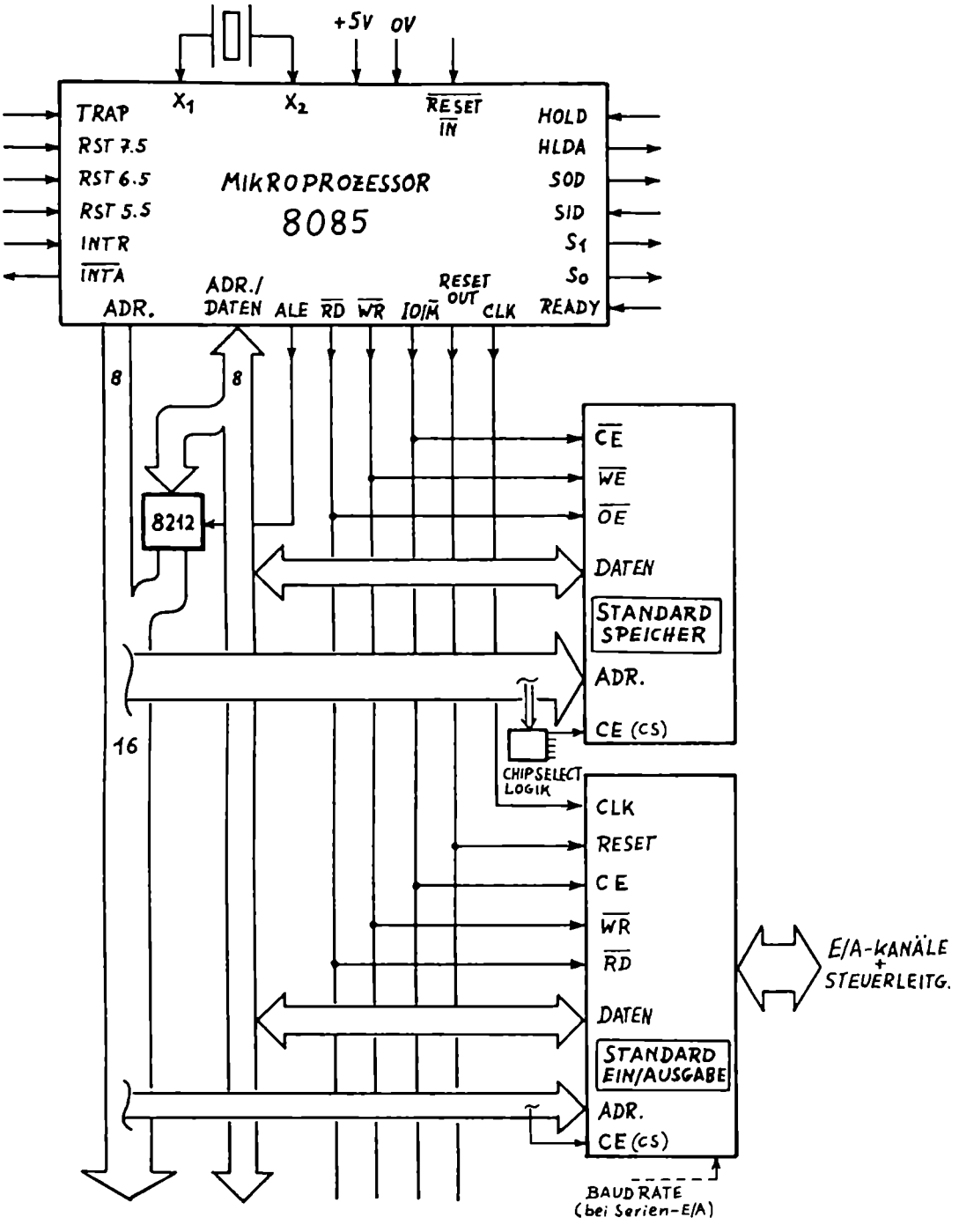


Abb.20: Mikrocomputer 8085 mit Standard Speicher- und E/A-Bausteinen

Als Standard-Speicherbausteine in Abb. 20 kann man sich z.B. die Bausteine 2716 (EPROM) oder 6116 (RAM) von Kapitel 5 eingesetzt denken, als Standard-E/A-Bausteine die früheren Bausteine 8255 (PIO) oder 8251 (USART). Ist der Gesamtspeicher wie üblich aus mehreren Bausteinen zusammengesetzt, so müssen zur Baustein-Auswahl wieder zusätzliche Adressleitungen herangezogen werden (wie in Kapitel 5 gezeigt). Bei mehreren E/A-Bausteinen wird wieder am einfachsten die "lineare Auswahl" von Kapitel 5.32 angewendet.

2.) Will man nur die Standard-E/A-Bausteine im System 8085 verwenden, so benötigt man hierzu nicht den vollständigen Adressbus A_0-A_{15}. Es genügt der Adressbus A_8-A_{15} des 8085! Um dies zu verstehen, muß man sich ins Gedächtnis zurückrufen, daß für die Adressierung von E/A-Bausteinen nur 8-Bit-Adressen verwendet werden und daß beim 8080/8085 System diese 8-Bit-Kanalnummern sowohl auf den oberen als auch auf den unteren acht Adressbits gleichlautend ausgegeben werden. Der Adressbus A_8-A_{15} des 8085 enthält also bereits die vollständige Kanalnummer.

3.) Für die Adressierung der E/A-Bausteine ist wieder sowohl die "isolierte Ein/Ausgabe" (Standard-E/A) als auch die "Speicher-Ein/Ausgabe" (Memory-Mapped-E/A) möglich (vgl. § 5.32). Memory-Mapped-E/A ist beim System 8085 sogar einfacher als beim System 8080, weil $\overline{RD}$ und $\overline{WR}$ alle Bausteine gleichermaßen bedient, also die auswählende A_{15}-Adressleitung (0 für Speicher, 1 für E/A) unmittelbar an die Chip-Select-Eingänge gelegt werden kann.

11.2) Ein Minimalsystem mit dem Mikroprozessor 8085 unter Verwendung von kombinierten Speicher-E/A-Bausteinen

Mit den bereits erwähnten, speziell für das System 8085 entwickelten hochintegrierten Speicher-E/A-Bausteinen ist es möglich, einen leistungsfähigen Mikrocomputer aus nur 3 Bausteinen aufzubauen. Diese neuen kombinierten Bausteine sind unmittelbar an das 8085-Bussystem anschließbar, da der Multiplex-Adress/Datenbus intern in diesen Bausteinen demultiplext wird (auf Veranlassung des Steuersignals ALE). Zunächst sollen die Speicher-E/A-Bausteine 8355/8755 und 8155/8156 vorgestellt werden und dann die Schaltung des 3-Chip-Minimalsystems diskutiert werden.

11.21) Der ROM-E/A-Baustein 8355 und der EPROM-E/A-Baustein 8755

Der Baustein 8355 ist ein kombinierter ROM-Ein/Ausgabe-Baustein, der Baustein 8755 die hierzu pinkompatible EPROM-Version. Die ersten Programmierversuche oder Einzelprogramme werden günstiger mit dem EPROM-E/A-Baustein 8755 durchgeführt. Die Bausteine enthalten folgende Funktionen (Abb. 21):

- Festwertspeicher mit 2K Byte Speicherzellen
- zwei 8-Bit-E/A-Kanäle

Die Speicher-Einheit enthält 2048 x 8 Speicherelemente (2K Byte), sie sind beim Baustein 8355 maskenprogrammiert, beim Baustein 8755 vom Anwender selbst programmierbar und UV-löschbar. Die Adressierung der 2K Speicherplätze erfolgt über 11 Leitungen (AD_0-AD_7,A_8-A_{10}), die Lese-Ansteuerung des Speichers geschieht über die Signale $\overline{RD}$ = 0 und IO/$\overline{M}$ = 0 (nur lesen!).

Die Ein/Ausgabe-Einheit besteht aus zwei 8-Bit-E/A-Kanälen. In beiden Kanälen kann jede einzelne Leitung indviduell als Ein- oder Ausgang programmiert werden, so daß der Anwender jede Kombination von Eingangs- und Ausgangsleitungen - von 16 Eingängen bis 16 Ausgängen - wählen kann. Hierzu besitzt jeder der beiden Kanäle A und B ein 8-Bit-Steuerregister für die bitweise Vorwahl

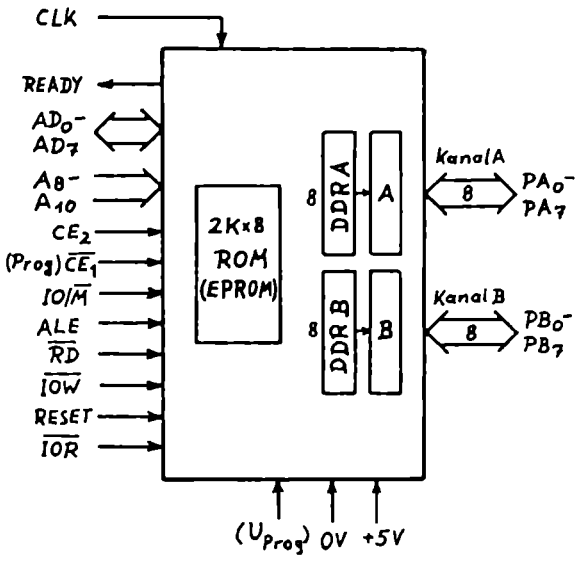


Abb.21: Blockschaltbild des ROM-E/A-Bausteins 8355 bzw. des EPROM-E/A-Bausteins 8755

Anschlußbezeichnungen		
PA_0-PA_7	:	Kanal A
PB_0-PB_7	:	Kanal B
CLK	:	Takt
READY	:	Bereit-Signal
AD_0-AD_7	:	Adress/Datenbus
A_8-A_{10}	:	Adressen
$\overline{CE}_1$,CE_2	:	Baustein-Freigabe
IO/$\overline{M}$	:	EA/Speicher-Auswahl
ALE	:	Adressenübernahme
$\overline{RD}$	:	Lesen
$\overline{IOW}$	:	EA-Schreiben
$\overline{IOR}$	:	EA-Lesen
RESET	:	Rücksetzen
U_{Prog}	:	Programmierspannung 8755
Prog	:	Programmiereingang 8755

der Datenflußrichtung (Data Direction Register: DDR). Eine 1 im Steuerregister verwandelt die zugehörige E/A-Leitung in eine Ausgabeleitung, eine 0 in eine Eingabeleitung. Jedes Steuerregister kann durch einen Schreib-Befehl ("OUT nr") mit dem gewünschten Bitmuster geladen werden (die Steuerregister können jedoch nicht gelesen werden).

Die Adressierung der beiden Kanäle A,B und der beiden Steuerregister DDR A, DDR B erfolgt durch nachstehende Adressen (Kanalnummern), wobei lediglich die 2 niedrigsten Bits der Adresse relevant sind:

E/A-Adresse

x x x x x x 0 0	:	Kanal A
x x x x x x 0 1	:	Kanal B
x x x x x x 1 0	:	DDR A
x x x x x x 1 1	:	DDR B

Die Abb. 21 zeigt das Blockschaltbild des Bausteins 8355/8755 mit den 40 Anschlüssen. Mit den Eingängen $\overline{IOR}$ und $\overline{IOW}$ ist es

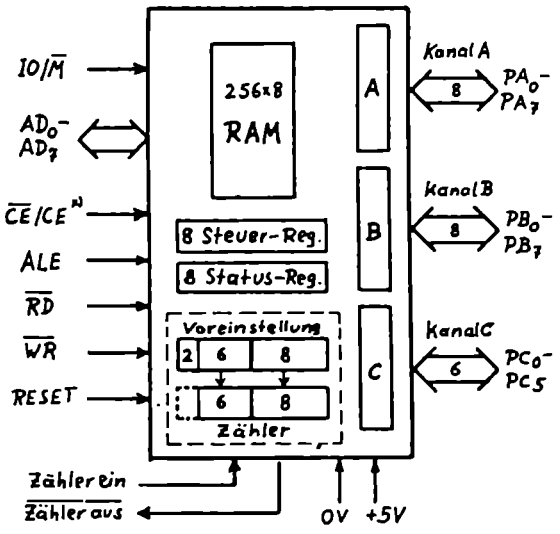
möglich, den E/A-Teil des Bausteins 8355/8755 unabhängig vom Zustand des Signals IO/$\overline{M}$ anzusteuern. Mit den 2 Freigabeeingängen $\overline{CE}_1$ und CE_2 (Chip Enable) wird der gesamte Baustein aktiviert, d.h. ansprechbar bzw. nichtansprechbar gemacht; dies wird bei Systemen mit mehreren Bausteinen zur Baustein-Auswahl (Chip Select) verwendet.

11.22) Der RAM-E/A-Zähler-Baustein 8155 bzw. 8156

Der Baustein 8155 bzw. 8156 enthält folgende Funktionen:

- Schreib-Lesespeicher RAM mit 256 x 8 Bit (256 Byte) Speicherzellen
- zwei 8-Bit-E/A-Kanäle und einen 6-Bit-E/A-Kanal
- einen programmierbaren 14-Bit-Zähler

Die beiden Versionen 8155 und 8156 sind identisch bis auf den Freigabeeingang (Chip Enable), der beim 8156 High-aktiv (CE), beim 8155 Low-aktiv ($\overline{CE}$) ist. Abb. 22 zeigt das Blockschaltbild des Bausteins mit den 40 Anschlüssen.



Anschlußbezeichnungen	
PA_0-PA_7	Kanal A
PB_0-PB_7	Kanal B
PC_0-PC_5	Kanal C
AD_0-AD_7	Adress/Datenbus
$\overline{CE}$/CE	Baustein-Freigabe
IO/$\overline{M}$	EA/Speicher-Auswahl
ALE	Adressenübernahme
$\overline{RD}$	Lesen
$\overline{WR}$	Schreiben
RESET	Rücksetzen
Zähler ein	
$\overline{\text{Zähler aus}}$	

Abb.22: Blockschaltbild des RAM-E/A-Zähler-Bausteins 8155/8156
(+) $\overline{CE}$ beim 8155, CE beim 8156)

Die Speicher-Einheit enthält ein 256 x 8 Bit statisches RAM, das für kleinere Systeme meist ausreichend ist. Die 256 Speicherplätze werden über 8 Leitungen (AD_0-AD_7) adressiert. Die Lese- bzw. Schreib-Ansteuerung des Speichers erfolgt über die Signale $\overline{RD}$ bzw. $\overline{WR}$, wobei der Eingang IO/$\overline{M}$ = O sein muß.

Die Ein/Ausgabe-Einheit besitzt zwei 8-Bit-Kanäle A und B, die als Ganzes entweder als Eingabe- oder als Ausgabekanal programmiert werden können (es können hier nicht einzelne Leitungen als Ein- oder Ausgang programmiert werden). Weiterhin ist ein 6-Bit-Kanal C vorhanden, der entweder als ein normaler Ein- oder Ausgabekanal verwendet werden kann, oder der aufgeteilt wird in Steuerleitungen für die Kanäle A und B, so daß Quittungsbetrieb ("handshake") möglich ist. Die Betriebsart, in welcher die Kanäle A,B und C arbeiten sollen, muß in einem Steuerregister niedergelegt werden, das durch Eingabe eines Steuerwortes programmiert wird (siehe später). Die 3 Kanäle werden über die nachstehenden Adressen angesprochen, wobei nur die 3 letzten Bits relevant sind:

E/A-Adresse

x x x x x O O 1 : Kanal A
x x x x x O 1 O : Kanal B
x x x x x O 1 1 : Kanal C

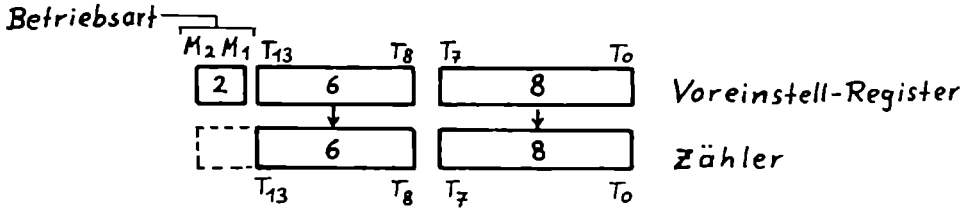
(Dabei muß der Eingang IO/$\overline{M}$ = 1 sein)

Die Zähler/Zeitgeber-Einheit besteht aus einem 14-Bit-Abwärtszähler, der von einem voreingestellten, programmierbaren Anfangszählerstand ausgehend, abwärts bis Null zählt. Bei diesem 14-Bit-Zähler können also Zahlen zwischen O und 2^{14} (dezimal 16 384) voreingestellt werden. An den Eingang des Zählers kann entweder das Taktsignal CLK des Mikroprozessors 8085 oder ein beliebiger externer Takt angelegt werden. Am Ausgang des Zählers wird entweder eine einzelne Rechteckwelle während der Zeitdauer des Abwärtszählens abgegeben (erste Hälfte der Zählzeit High, zweite Hälfte Low: ‾‾‾|___), oder aber ein einzelner negativer (Low) Impuls bei Erreichen der Null (‾‾‾‾|_|)[+]. Da der Zähler auch repetierend betrieben werden kann (also immer wieder nach Errei-

[+] Impulsdauer gleich der Periode des Eingangssignales

chen der Null auf den voreingestellten Anfangswert zurückspringt), können bei dieser Betriebsart am Ausgang entweder periodische Rechteckschwingungen (⎍⎍⎍⎍) oder aber eine Folge negativer Impulse (⎍‾⎍‾⎍‾⎍‾) abgenommen werden. Die gewünschte Betriebsart des Zählers wird vom Anwender einprogrammiert. Das Starten und Stoppen des Zählers wird durch das Steuerregister veranlaßt (siehe später).

Die 14 Bits des Zählers sind in 2 Gruppen zu 8 Bit (T_0- T_7) und 6 Bit (T_8- T_{13}) unterteilt. Vor den Zähler ist ein 2-Byte-Voreinstell-Register (Count Length Register) vorgeschaltet.



In dieses Voreinstellregister wird der gewünschte Anfangsstand T_0- T_{13} des Zählers eingeschrieben. Die beiden noch freien höchsten Bits D_{14} und D_{15} des Voreinstellregisters werden zur Festlegung der beschriebenen 4 Betriebsarten (Moden) des Zählers wie folgt benützt:

D_{15} M_2	D_{14} M_1	Betriebsart	Ausgangssignal des Zählers
0	0	einmaliges Abwärtszählen	einzelne Rechteckwelle
0	1	repetierend	Rechteckschwingung
1	0	einmaliges Abwärtszählen	einzelner negativer Impuls (bei Nulldurchgang)
1	1	repetierend	negative Impulsfolge

Die 2 Bytes des Voreinstell-Registers haben folgende E/A-Adressen:

E/A-Adresse

x x x x x 1 0 0 : niederes Byte des Voreinstellregisters (T_0-T_7)

x x x x x 1 0 1 : höheres Byte des Voreinstellregisters samt Betriebsarten (T_8- T_{13}, M_1 , M_2)

(Bei der Adressierung ist IO/$\overline{M}$ = 1).

Das Steuerregister (Command Register) im Baustein 8155/8156 dient zur Festlegung des Betriebszustandes der 3 E/A-Kanäle und zum Starten und Stoppen des Zählers. Die einzelnen Bits in dem 8-Bit-Steuerregister (D_7- D_0) haben folgende Bedeutung:

D_0: Betriebszustand von Kanal A
D_1: Betriebszustand von Kanal B
→ 0 = Eingabekanal, 1 = Ausgabekanal

D_2, D_3: definiert Betriebszustand von Kanal C
- 00: 6 Eingänge
- 11: 6 Ausgänge
- 01: 3 Steuer-Leitungen, 3 Ausgänge
- 10: 2 x 3 Steuer-Leitungen

D_4: Unterbrechungsfreigabe für Kanal A
D_5: Unterbrechungsfreigabe für Kanal B
→ 1 = freigeben, 0 = sperren

D_6, D_7: definiert Starten und Stoppen des Zählers
- 00: NOP (Leerbefehl)
- 01: STOP sofort
- 10: STOP beim nächsten Nulldurchgang
- 11: START

Der Inhalt des Steuerregisters wird über folgende E/A-Adresse eingeschrieben:

E/A-Adresse

x x x x x 0 0 0 : Steuerregister

(Bei der Adressierung ist IO/$\overline{M}$ = 1).

Ergänzung:
a) In das Steuerregister kann nur eingeschrieben werden! Man kann daher die gleiche Adresse xxxxx000 dazu benützen, um ein weiteres Register des Bausteins 8155/8156, das "Status-Register", nur zu lesen. Das Statusregister gibt Auskunft über den momentanen Zustand der Kanäle A und B und des Zählers.
b) Auch in die 2 Voreinstell-Register kann nur eingeschrieben werden! Die gleichen Adressen (xxxxx100 und xxxxx101) können daher zum Lesen des momentanen Zustands der 14 Bit des Zählers benützt werden.

11.23) Die ausgeführte Schaltung des Minimal-Systems

In dem Blockschaltbild Abb. 23 wird eine Schaltung angegeben, in welcher der Mikroprozessor 8085 mit dem kombinierten EPROM-E/A-Baustein 8755 (bzw. dem entsprechenden ROM-E/A-Baustein 8355) und dem kombinierten RAM-E/A-Zähler-Baustein 8156 zu einem leistungsfähigen, aus nur 3 Bausteinen bestehenden Minimalsystem

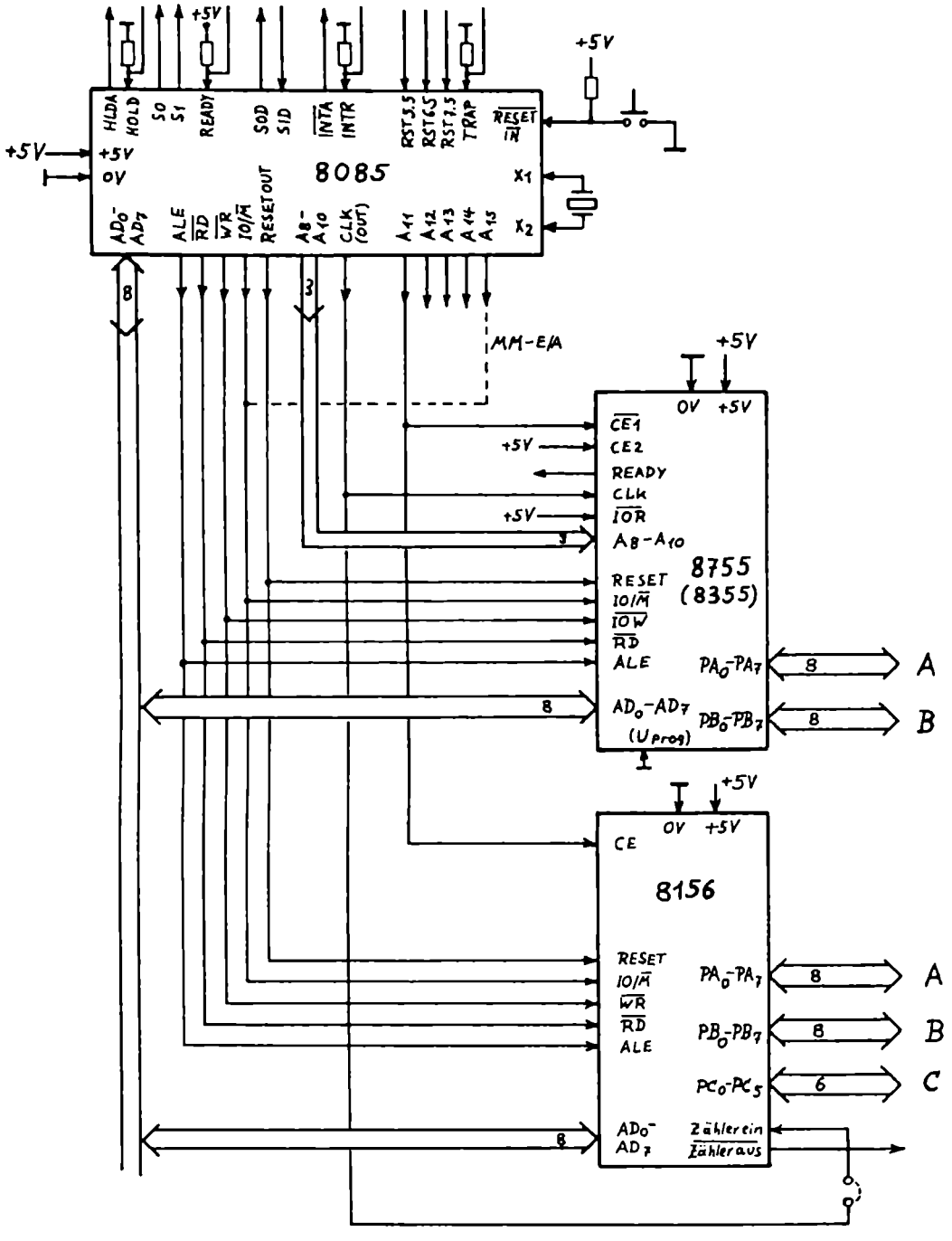


Abb. 23: Ein 3-Chip-Minimalsystem mit dem Mikroprozessor 8085 und den kombinierten Speicher-E/A-Bausteinen 8755 und 8156

zusammengeschaltet ist. Dieses 3-Chip-Mikrocomputersystem hat folgende Leistungsmerkmale:

- 2K Byte Programmspeicher (EPROM oder ROM)
- 256 Byte Schreib-Lese-Speicher (RAM)
- 4 x 8 und 1 x 6 programmierbare Ein/Ausgabe-Leitungen
- 1 programmierbarer 14-Bit-Zähler/Zeitgeber
- 1 serielle Ein/Ausgabe-Schnittstelle
- 5 Interrupt-Leitungen
- 1,3 µs Befehlszykluszeit
- einfache +5V Spannungsversorgung

Die Schaltung von Abb. 23 ist nach der Kenntnis der Einzelbausteine unmittelbar verständlich. Die Adresse A_{11} wirkt bei dieser Schaltung als Chip-Select-Signal, d.h. $A_{11} = 0$ aktiviert den Baustein 8755 (über den Eingang $\overline{CE}_1$), $A_{11} = 1$ aktiviert den Baustein 8156 (über den Eingang CE). Die Schaltung arbeitet mit isolierter Ein/Ausgabe. (Soll das Minimalsystem mit Memory-Mapped-Ein/Ausgabe arbeiten, so wird die Adreßleitung A_{15} anstelle der IO/$\overline{M}$-Leitung des Mikroprozessors 8085 verwendet (gestrichelte Leitung in Abb. 23)).

11.24) Weitere Peripheriebausteine des Systems 8080/8085

Es soll zum Abschluß noch auf einige Erweiterungsmöglichkeiten des beschriebenen Mikrocomputersystems kurz hingewiesen werden.

Für das Computersystem 8080/8085 wurde eine Reihe von hochintegrierten, programmierbaren Peripheriebausteinen entwickelt, die sehr komplexe Aufgaben übernehmen können und bei größeren Mikrocomputersystemen den Mikroprozessor von programmintensiven Steuer- und Rechenfunktionen entlasten können. Es sind dies zum Beispiel folgende Bausteine:

- programmierbarer "Floppy-Disk-Controller 8271 und 8272" zum Anschluß von 1 bis 4 Disketten-Laufwerken an einen 8-Bit-Mikrocomputer.
- programmierbarer "CRT-Controller 8275 und 8276" zum Anschluß eines Bildschirmmonitors mit zeilenförmiger Abrasterung.

- programmierbares "Tastenfeld/Anzeige-Interface 8279" zur Abtastung eines Tastenfeldes von 64 Kontakten und zur 16-stelligen numerischen Anzeige.

- programmierbarer "DMA-Controller 8237 und 8257" für die Steuerung des direkten Speicherzugriffs durch Peripheriegeräte.

- "Universelles Peripherie-Interface (UPI) 8041/8741 und 8042/8742" mit eigenem vollständigem 8-Bit-Mikrocomputer für den Einsatz als Controller in den verschiedensten Anwendungsfällen wie Tastenfeld-Scanning, Drucker-Steuerung, Multiplexen von Zeilen-Displays usw.

- "Arithmetik-Prozessor 8231" (16/32 Bit Genauigkeit) mit folgenden Funktionen: Addition, Subtraktion, Multiplikation, Division, Trigonometrische Funktionen, Quadratwurzel, Logarithmus, Exponentialfunktionen. Fest- und Gleitkomma-Arithmetik.

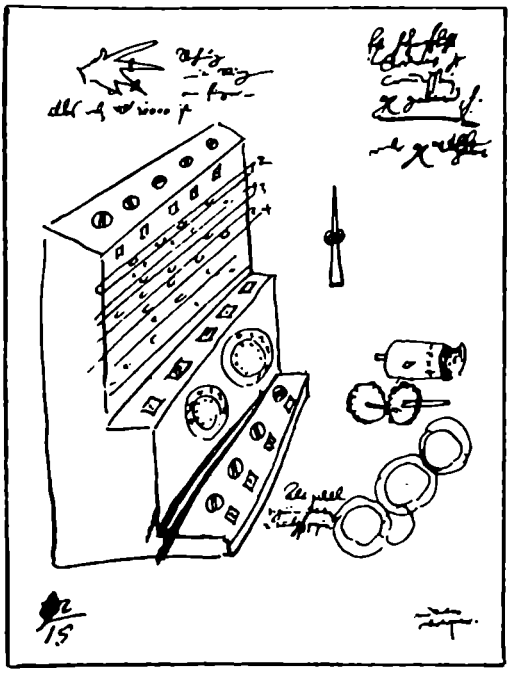
- "Gleitkomma-Prozessor 8232" (32/64 Bit Genauigkeit) mit folgenden Funktionen: Addition, Subtraktion, Multiplikation, Division.

- programmierbarer "Multifunktionsbaustein 8256" (MUART: Microprocessor Universal Asynchronous Receiver Transmitter) mit 1 programmierbaren seriellen und 2 programmierbaren 8-Bit-parallelen Datenschnittstellen, sowie 5 unabhängigen, programmierbaren 8-Bit-Zählern und einer Unterbrechungssteuerung mit 8 Prioritätsebenen.

- programmierbarer "Interrupt-Controller (PIC) 8259" zur Erzeugung von 8 Vektor-Interrupts mit Prioritäts-Rangfolge. Kaskadierbar bis zu einer Prioritätskette von maximal 64 Vektor-Interrupts.

- programmierbarer "Intervall-Timer 8253 und 8254" mit drei 16-Bit-Zählern zur Lösung von Timing-Problemen in Mikrocomputern.

Bezüglich der genauen Beschreibung dieser sehr leistungsfähigen peripheren Bausteine wird auf die Datenbücher der Herstellerfirmen verwiesen.



Schickard's Skizze
seiner Rechenmaschine
(1623)

****
12) DER MIKROPROZESSOR Z80
****

12.1) Einführung

In den folgenden Kapiteln soll ein weiterer 8-Bit-Mikroprozessor vorgestellt werden, der eine Weiterentwicklung des Prinzips des Mikroprozessors 8080 darstellt und der von der Fa. ZILOG seit 1976 auf dem 8-Bit-Markt angeboten wird: der Mikroprozessor Z80. Seine Funktionsweise wird aus der Kenntnis des 8080 unmittelbar verständlich. (Weitere Herstellerfirmen siehe S.23)

Der Mikroprozessor Z80 ist gegenüber dem Mikroprozessor 8080 höher integriert und leistungsfähiger: Er enthält (bis auf einen einfachen Taktgenerator) die gesamte CPU auf einem einzigen Chip integriert (Abb. 24), wobei zusätzliche Funktionen und zusätzliche Baugruppen in die CPU mit aufgenommen wurden. Ein Vergleich mit dem Mikroprozessor 8080 macht die höhere Leistungsfähigkeit des Mikroprozessors Z80 deutlich.

Äußerlich ist der Mikroprozessor Z80 - wie der 8080 - durch einen 16-Bit-Adressbus und einen 8-Bit-Datenbus gekennzeichnet. Die 4 wichtigsten Steuerleitungen haben jedoch beim Z80 andere Funktionen; es bedeutet

$\overline{RD}$: das allgemeine Lesesignal (für Speicher und Eingabe/Ausgabe)
$\overline{WR}$: das allgemeine Schreibsignal (für Speicher und Eingabe/Ausgabe)
$\overline{MREQ}$: das aktivierende Steuersignal für die Speicherbausteine
$\overline{IORQ}$: das aktivierende Steuersignal für die Eingabe/Ausgabe-Bausteine
(Diese 4 Steuersignale sind Low-aktiv)

Im Unterschied zum 8080 benötigt der Z80 nur ein einziges rechteckförmiges Taktsignal, das in einem getrennten Generator erzeugt wird.

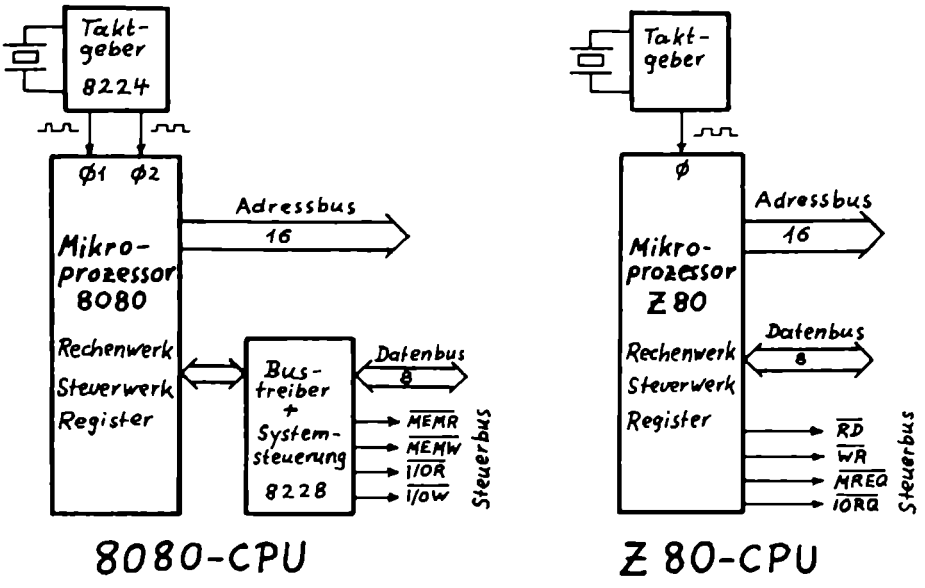


Abb. 24: Gegenüberstellung der Zentraleinheit des Systems 8080 und des Systems Z80
(es sind nur die wichtigsten Leitungen eingezeichnet)

Das Rechenwerk (ALU) des Z80 enthält die bekannten 8080-Funktionen in teilweise erweiterter Form: 8-Bit-arithmetische und 8-Bit-logische Operationen, erweiterte 16-Bit-arithmetische Operationen, erweiterte Rotations- und Schiebeoperationen. Daneben enthält das Rechenwerk des Z80 die Möglichkeit für zusätzliche Einzelbit-Operationen: einzelne Bits können gesetzt, rückgesetzt oder getestet werden.

Der Registersatz des Z80: Besonders augenfällig ist die Vergrößerung der Zahl der Register beim Z80 (Abb. 25). Gegenüber dem 8080 enthält der Mikroprozessor Z80 folgende zusätzliche statische Register:
1.) Einen vollständigen zweiten Satz von universellen 8-Bit-Registern A',B',C',D',E',H',L' und ein zweites Flag-Register F'. Diese Register sind also beim Z80 doppelt vorhanden. Durch einfache 1-Byte-Austauschbefehle (exchange) können die Registerinhalte ausgetauscht werden. Man kann dadurch sehr schnell von

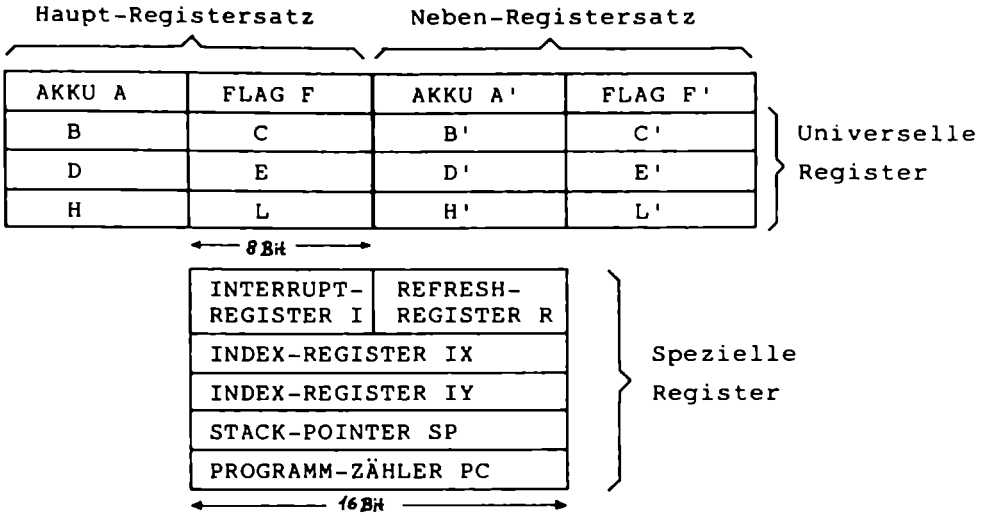


Abb. 25: Der Registersatz des Mikroprozessors Z80

einem Hauptprogramm in ein Nebenprogramm wechseln (etwa bei schnellen Interrupt-Anforderungen), da z.B. mit nur einem einzigen 1-Byte-Befehl (EXX) die Inhalte der 6 Register B - L in den zweiten Registersatz B' - L' innerhalb der CPU gerettet werden können, ohne daß man den zeitraubenden Umweg über den externen Arbeits-Speicher zur Rettung der Registerinhalte zu machen braucht.

2.) Zwei 16-Bit-Index-Register IX und IY. In diesen unabhängigen 16-Bit-Registern können "Basis-Adressen" abgelegt werden, die für die "indizierte Adressierung" von Speicherplätzen benötigt werden. Durch diesen neuen Adressierungs-Modus, die "Index-Adressierung", kann ein Speicherplatz relativ zu einer Basis-Adresse aufgerufen werden. (Man kann also einen Speicherplatz lesen oder beschreiben, der um eine bestimmte Zahl von Plätzen - die "Distanzzahl" - von der Basis-Adresse entfernt ist.) Dieser Adressierungs-Modus vereinfacht solche Programme erheblich, bei denen z.B. Daten-Tabellen benützt werden (siehe auch später im Befehlssatz).

3.) Ein 8-Bit-Interrupt-Register I, das den indirekten Aufruf von Unterbrechungs-Service-Routinen ermöglicht, die an beliebiger Stelle im Programmspeicher angeordnet sein können. Dem Interrupt-

Register I werden dabei die höheren 8 Bit einer Zeiger-Adresse eingegeben, während die niederen 8 Bit dieser Zeiger-Adresse aus dem peripheren Gerät kommen, das den Interrupt anfordert (genaueres später).
4.) Ein 8-Bit-Refresh-Register R, das die unmittelbare Verwendung von dynamischen RAM-Speicherbausteinen ohne zusätzliche Refresh-Hardware ermöglicht.

Die Interrupt-Möglichkeiten wurden beim Mikroprozessor Z80 gegenüber dem 8080 erweitert. Ein zusätzlicher nicht-maskierbarer Interrupt-Eingang $\overline{NMI}$ steht zur Verfügung und unter Benützung des Interrupt-Eingangs $\overline{INT}$ sind 3 verschiedene Unterbrechungs-Moden programmierbar, wobei einer davon der bekannte Interrupt-Modus des Mikroprozessors 8080 ist.

Software

Der wohl auffälligste Unterschied zwischen den Mikroprozessoren Z80 und 8080 ist der beträchtlich erweiterte Befehlssatz des Z80. Während der Befehlssatz des 8080 244 Befehle umfaßt, stehen beim Mikroprozessor Z80 insgesamt 694 *) Befehle zur Verfügung. Da mit 8-Bit-Operationscodes insgesamt nur $2^8 = 256$ verschiedene Befehle möglich sind, war man beim Z80 gezwungen, bei den hinzukommenden Befehlen größtenteils auf Operationscodes mit 2 Byte Länge überzugehen!
Die 450 neu hinzugekommenen Befehle enthalten zum Teil ganz neue Klassen von Befehlen (z.B. Blocktransferbefehle, Zeichensuchbefehle, Einzelbitbefehle, Index-Adressierungen), zum andern wurden die uns schon bekannten Befehlsklassen ergänzt und vervollständigt. Der Befehlssatz des Mikroprozessors 8080 ist jedenfalls vollständig im Befehlssatz der Z80 enthalten, und zwar mit unverändertem Maschinencode, jedoch mit veränderten mnemonischen Abkürzungen. Im Maschinencode vorliegende Programme des Systems 8080 sind also auch im System Z80 lauffähig (der Z80 ist "aufwärtskompatibel" zum 8080).

---

*) neben diesen 694 vom Hersteller "autorisierten" Befehlen existiert noch eine große Zahl nicht autorisierter Befehle

Der Mikroprozessor Z80 wird in NMOS-Technologie hergestellt und in 4 Versionen angeboten *):

Z80 mit 2,5 MHz Taktfrequenz und 1,6 µs Befehlszykluszeit
Z80 A mit 4 MHz Taktfrequenz und 1,0 µs Befehlszykluszeit
Z80 B mit 6 MHz Taktfrequenz und 0,67µs Befehlszykluszeit
Z80 H mit 8 MHz Taktfrequenz und 0,5 µs Befehlszykluszeit

(Meist wird im folgenden vereinfachend von "dem" Z80 gesprochen)

Der Z80 benötigt nur eine einzige Versorgungsspannung +5 V.

Der Aufbau von Mikrocomputern mit dem Mikroprozessor Z80 als CPU kann unter Verwendung von herkömmlichen Standard-Speicherbausteinen und Standard-Ein/Ausgabe-Bausteinen erfolgen, wie wir sie schon beim System 8080 kennengelernt haben. Daneben werden von der Fa. ZILOG (u.a.) spezielle, auf das System Z80 zugeschnittene Ein/Ausgabe- und Peripheriebausteine angeboten, welche die Eigenschaften des Z80 (z.B. seine Interrupt-Möglichkeiten) optimal zur Geltung bringen.

Zusammenfassend seien noch einmal die charakteristischen Daten des Z80 aufgeführt:

- 8-Bit-Mikroprozessor in NMOS-Technologie im 40-Pin-Gehäuse
- 4 Versionen : Z80 (2,5 MHz), Z80 A (4 MHz), Z80 B (6 MHz), Z80 H (8 MHz) *)
- Doppelter Satz von universellen Registern und Flag-Registern
- 2 16-Bit-Index-Register
- Refresh-Hardware integriert, unmittelbare Verwendung von dynamischen RAM-Bausteinen möglich
- 2 Interrupt-Eingänge und 3 programmierbare Interrupt-Moden
- +5 V Betriebsspannung
- umfangreicher Befehlssatz mit 694 Befehlen
- Anschluß von Standard-Speicherbausteinen und Standard-Ein/Ausgabe-Bausteinen möglich
- spezielle Ein/Ausgabe-Bausteine (PIO, SIO etc.) vorhanden

---

*) Daneben werden auch verlustleistungsarme CMOS-Versionen des Z80 angeboten.

12.2) Der Aufbau des Mikroprozessors Z80

In Abb. 26 ist der interne Aufbau des Mikroprozessors Z80 im Blockdiagramm wiedergegeben. Beim Vergleich mit dem sehr ähnlichen Blockdiagramm des Mikroprozessors 8080 (Abb. 5, S. 49) fällt die größere Anzahl von verfügbaren Registern ins Auge. Im folgenden werden die gegenüber dem Mikroprozessor 8080 veränderten oder in die CPU des Mikroprozessors Z80 neu aufgenommenen Baugruppen noch einmal systematisch zusammengestellt, und es werden die Bedeutungen der 40 Anschlüsse des Z80-Gehäuses beschrieben.

Die Arithmetik-Logik-Einheit (ALU) des Z80 enthält neben den bekannten 8-Bit-Funktionen (Addition, Subtraktion, UND, ODER, EXCLUSIV-ODER, Vergleichen, Inkrementieren, Dekrementieren, Rotieren) zusätzliche Rotations- und Schiebefunktionen, sowie die zusätzlichen Einzelbit-Operationen: Bit setzen, Bit rücksetzen, Bit testen (im Befehlssatz des Mikroprozessors Z80 sind hierfür 240 zusätzliche Befehle für Einzelbit-Operationen enthalten!). Die arithmetischen (unechten) 16-Bit-Operationsmöglichkeiten wurden beim Z80 erweitert.

Der neu hinzugekommene zweite 8-Bit-Registersatz A',F',B'-L' dient dem raschen Austausch von Registerinhalten innerhalb der CPU. Es ist schon hier festzuhalten, daß dieser zweite Registersatz nur durch 2 Befehle angesprochen werden kann:
der Befehl EXX veranlaßt den Austausch der Registerinhalte B-L mit B'-L', der Befehl EX AF,AF' den Austausch der Registerinhalte A,F mit A',F'.

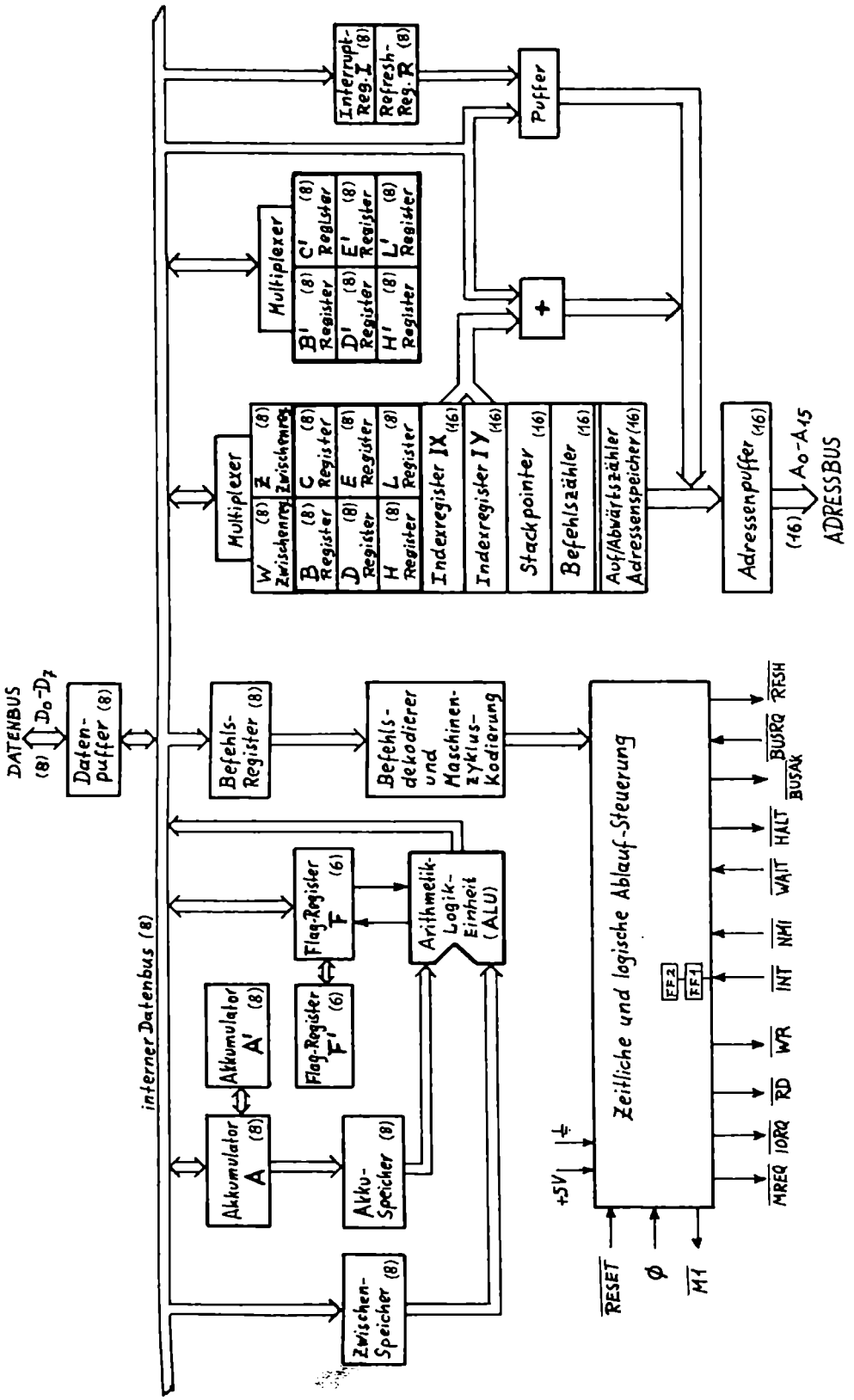
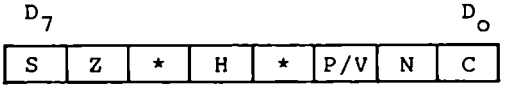


Abb.26: Blockdiagramm des internen Aufbaus des Mikroprozessors Z80

Die Flag-Register F und F' enthalten beim Z80 jeweils 6 Bedingungsbits (Flags), die durch verschiedene CPU-Operationen gesetzt oder rückgesetzt werden. Die Bezeichnungen der Flags sind:

S: Sign-Flag (Vorzeichenbit)
Z: Zero-Flag (Nullbit)
P/V: Parity-Flag (Paritätsbit) und gleichzeitig Overflow-Flag (Überlaufbit)
C: Carry-Flag (Übertragsbit)
H: Half-Carry-Flag (Hilfsübertragsbit, beim 8080 AC-Flag genannt)
N: Add/Subtract-Flag (Additions/Subtraktionsbit)

Gegenüber dem 8080 ist also beim Mikroprozessor Z80 das N-Flag hinzugekommen und das Paritäts-Flag hat eine Doppelfunktion. Einzeln abfragbar (zur Verwendung bei bedingten Jump-, CAll-, Return-Befehlen) sind nach wie vor nur die 4 Flags S,Z,P,C ! Das Format des Flag-Bytes lautet (in Übereinstimmung mit den früheren Festlegungen von S.51):



Die mit * bezeichneten Bits sind beim Z80 undefiniert.

Das neu hinzugekommene N-Flag ist ein "internes" Bit; es dient der CPU bei Dezimalausgleichs-Operationen (Befehl DAA) zum Hinweis, ob die zuletzt ausgeführte Operation eine Addition oder eine Subtraktion war. Bei allen Additionsbefehlen wird N = 0 gesetzt, bei allen Subtraktionsbefehlen N = 1.
Das P/V-Flag dient beim Z80 einem doppelten Zweck: Es zeigt bei logischen Operationen (UND etc.) die Parität an, wie einst beim 8080. Daneben signalisiert es bei arithmetischen Zweierkomplement-Operationen (siehe S.11) das Überschreiten des zulässigen Zahlenbereichs (-128 bzw. +127); in diesem Fall wird P/V = 1 gesetzt.

Die beiden neu hinzugekommenen 16-Bit-Index-Register IX und IY dienen, wie bereits erwähnt, unabhängig voneinander der Aufnahme je einer Basis-Adresse, die für die "indizierte Adressierung"

verwendet wird. Die bei indizierter Adressierung wirksam werdende Adresse (IX+d) bzw. (IY+d) ergibt sich aus der Addition einer 8-Bit-"Distanzzahl" d zur 16-Bit-Basisadresse (IX) bzw. (IY); hierfür ist in der Z80-CPU ein kleiner Addierer vorhanden.

Auf das 8-Bit-Interrupt-Register I wird später im Zusammenhang mit den Interrupt-Möglichkeiten des Z80 noch genauer eingegangen werden.

Das 8-Bit-Refresh-Register R wird für die Erzeugung von Refresh-Adressen zum Auffrischen von dynamischen RAM-Speicherbausteinen verwendet. Die Refresh-Adresse wird während jedes Befehlsablaufs automatisch im 1. Maschinenzyklus M1 erzeugt (siehe hierzu auch Abb. 28). Das Refresh-Register arbeitet dabei als 7-Bit-Zähler, der bei jedem ausgeführten Befehl automatisch um 1 hochgezählt wird und eine neue Refresh-Adresse bildet, die dann auf den 7 niederwertigen Leitungen A0 - A6 des Adressbusses während des 1. Maschinenzyklus ausgesandt wird. Mit minimalem Schaltungsaufwand kann somit der Refresh-Vorgang für die dynamischen Bausteine durchgeführt werden.

Taktsignal: Der Mikroprozessor Z80 benötigt im Unterschied zum 8080 nur ein 1-Phasen-Taktsignal (Clock). Dieses einfache Rechtecksignal wird extern aus einem Quarzoszillator mit nachfolgender Impulsformerschaltung (Taktgenerator) erzeugt und dem Ø-Eingang des Prozessors zugeführt.Bei geringeren Anforderungen an die Frequenzkonstanz des Taktsignales kann der Quarzoszillator durch einen LC- oder RC-Oszillator ersetzt werden. (Da der Z80 mit statischen Schaltungen arbeitet, ist die Taktfrequenz nach unten nicht begrenzt, im Unterschied zum 8080/85.)

Die 40 Anschlußleitungen, über die der Mikroprozessor Z80 mit dem übrigen System in Verbindung steht, sind in Abb.27 zusammengestellt (für die mit + bezeichneten Anschlüsse gibt es Entsprechungen beim Mikroprozessor 8080). Die Anschlüsse haben im einzelnen folgende Funktionen:

+ A_0- A_{15}:Ausgänge für den Adressbus mit 16 Leitungen
+ D_0- D_7 :Bidirektionaler Datenbus mit 8 Leitungen

Für den "Steuerbus" im engeren Sinn sind folgende 4 Anschlüsse vorhanden (Low-aktiv):

- $\overline{RD}$ (Read): Ausgang für das Lesesignal für Speicher und Ein/Ausgabe-Einheit
- $\overline{WR}$ (Write): Ausgang für das Schreibsignal für Speicher und Ein/Ausgabe-Einheit
- $\overline{MREQ}$ (Memory Request): Ausgang für das Steuersignal für die Speicherbausteine
- $\overline{IORQ}$ (Input Output Request): Ausgang für das Steuersignal für die Ein/Ausgabe-Bausteine

Die beiden Interruptleitungen des Z80 sind:

- $\overline{NMI}$ (Non Maskable Interrupt): Eingang für eine nicht-maskierbare Interrupt-Anforderung
+) $\overline{INT}$ (Interrupt): Eingang für eine maskierbare Interrupt-Anforderung (in 3 verschiedenen Moden programmierbar)

Die restlichen Anschlüsse haben folgende Bedeutung:

- $\overline{M1}$: Der Ausgang $\overline{M1}$ zeigt durch Low-Signal an, daß gerade ein M1-Maschinenzyklus ausgeführt wird. Ist $\overline{M1}$ gleichzeitig mit $\overline{IORQ}$ im Low-Zustand, so liegt ein Interrupt-Quittungssignal vor. Das Interrupt-Quittungssignal (das dem $\overline{INTA}$ des 8080 entspricht) wird also durch ODERieren von $\overline{M1}$ und $\overline{IORQ}$ erzeugt.
+ $\overline{RESET}$: Eingang für das Rücksetzsignal.Ein Low-Signal auf dieser Leitung bewirkt:
  Programmzähler PC auf 0000 gesetzt
  Register I und R auf 00 gesetzt
  Interrupt-Modus 0 eingestellt
  Interrupt gesperrt (DI)
+) $\overline{WAIT}$: Ein Low-Signal an diesem Eingang löst einen Wartezustand des Prozessors aus (vgl. RDYIN beim 8080)
+) $\overline{BUSRQ}$ (Bus Request): Dieser Eingang ermöglicht direkten Speicherzugriff (DMA). Adressbus, Datenbus und die 4 engeren Steuerbusleitungen werden in den hochohmigen Tri-State-Zustand gebracht, solange $\overline{BUSRQ}$ = 0 ist (vgl. HOLD beim 8080/85).

+) $\overline{\text{BUSAK}}$ (Bus Acknowledge): Dieser Ausgang gibt eine Quittung für die $\overline{\text{BUSRQ}}$-Anforderung. Low-Signal zeigt dem peripheren Gerät an, daß Daten-, Adress- und Steuerbus im hochohmigen Zustand sind (vgl. HLDA beim 8080/85).

- $\overline{\text{RFSH}}$ (Refresh): Low-Signal auf diesem Ausgang zeigt an, daß die Adressleitungen A0 - A6 eine Refresh-Adresse enthalten (A7 ist 0).
- $\overline{\text{HALT}}$: Dieser Ausgang zeigt an, daß der Prozessor einen Software-HALT-Befehl erhalten hat und in diesem HALT-Zustand verharrt.
- Ø: TTL-Eingang für das rechteckförmige Taktsignal (Clock)
- +5V,0V: Versorgungsspannung

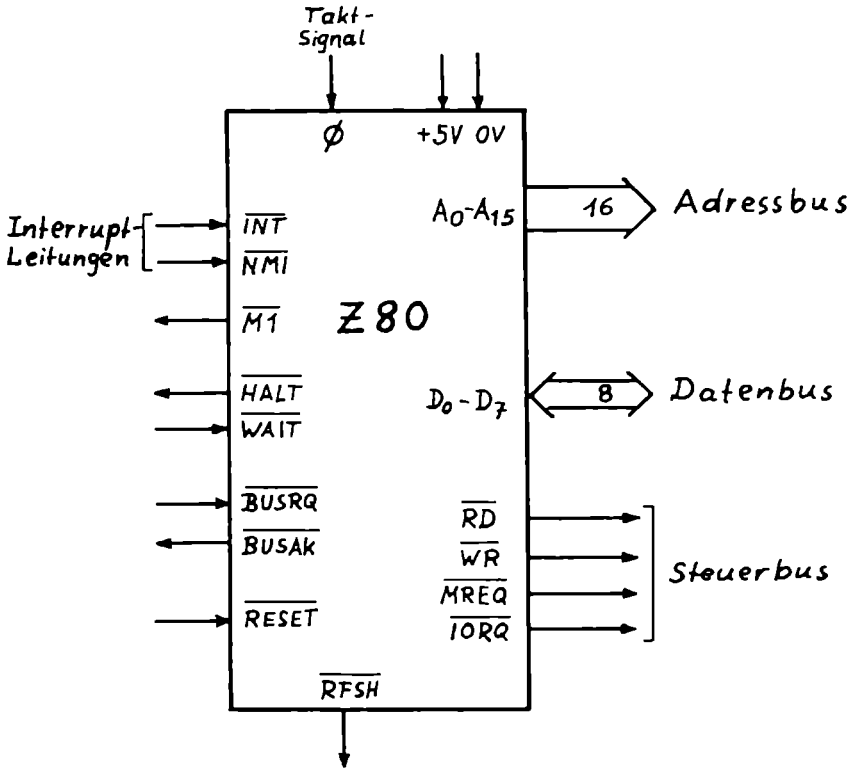


Abb. 27: Die 40 Anschluß-Leitungen des Mikroprozessors Z80

12.3) Zeitlicher Ablauf der Signale beim Mikroprozessor Z80

Wie bei allen Mikroprozessoren, so wird auch beim Z80 der zeitliche Ablauf der internen Prozesse durch das Taktsignal (Clock) bestimmt und synchronisiert. Das Taktsignal ist beim Mikroprozessor Z80 ein einfaches, einphasiges Rechtecksignal. Die Frequenz des Taktsignales, die zugehörige Taktdauer und die (aus 4 Takten bestehende) kürzeste Befehlszyklus-Dauer ist im folgenden für die 4 unterschiedlich schnellen Versionen des Mikroprozessors Z80 zusammengestellt:

Prozessor	Taktfrequenz	Taktperiode	kürzester Befehlszyklus
Z80	2,5 MHz	0,4 µs	1,6 µs
Z80 A	4 MHz	0,25 µs	1,0 µs
Z80 B	6 MHz	0,16 µs	0,6 µs
Z80 H	8 MHz	0.125 µs	0,5 µs

In Abb. 28 ist der zeitliche Ablauf der Signale auf verschiedenen Leitungen des Mikroprozessors Z80 dargestellt, wobei als Demonstrations-Beispiel die Einholung und Ausführung des Befehls " OUT (n),A " gewählt wurde. Diesem Befehl entspricht beim Mikroprozessor 8080 der Befehl " OUT nr ", der in seinem zeitlichen Ablauf in Abb. 16 auf Seite 137 dargestellt wurde.

Zunächst soll hier der zeitliche Ablauf des Befehls OUT (n),A im Grundsatz erklärt werden:
Im 1. Maschinenzyklus M1 wird der Operationscode des Befehls aus dem Programspeicher eingeholt, die hierzu benötigte Speicheradresse (der Inhalt des Befehlszählers) wird von der CPU auf den Adressbus gegeben. Der Operationscode (Byte 1 des Befehls) wird vom Speicher auf den Datenbus gegeben und von der CPU ins Befehlsregister übernommen.
Im 2. Maschinenzyklus M2 wird auf gleiche Weise das 2. Byte des Befehls (die 8-Bit-Kanalnummer) auf den Datenbus gegeben und in die CPU geholt, die hierzu vom Befehlszähler ausgegebene Adresse ist um 1 höher.

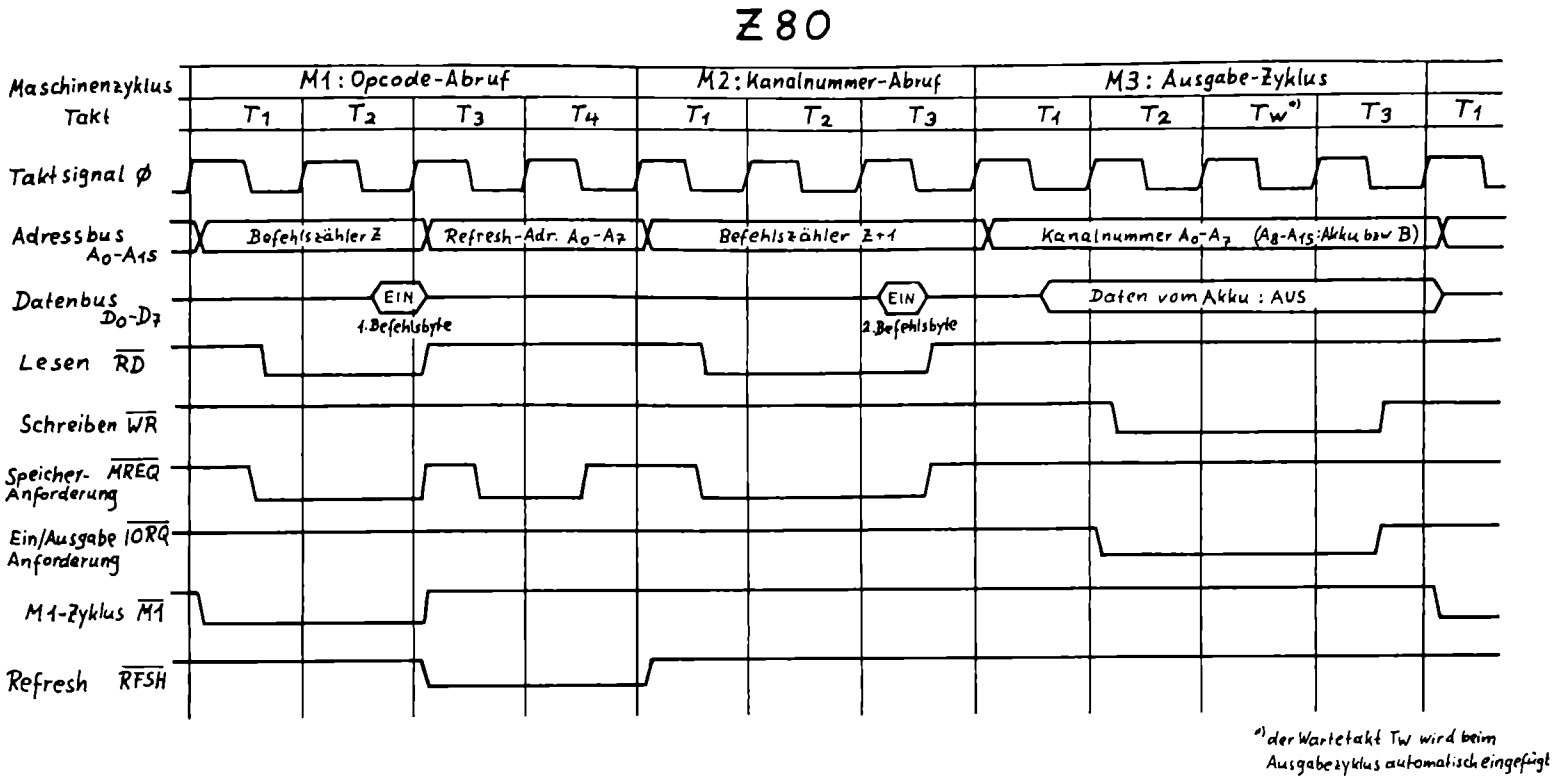


Abb.28: Zeitlicher Ablauf der Signale des Mikroprozessors Z80 beim Ausgabebefehl " OUT (n),A "

Im 3. Maschinenzyklus M3 gibt die CPU die 8-Bit-Kanalnummer auf die niederwertigen 8 Adressleitungen A0 - A7 und das im Akkumulator enthaltene Datenwort wird anschließend über den Datenbus an den adressierten E/A-Kanal ausgegeben.
Abb. 28 zeigt das zeitliche Zusammenspiel zwischen den Signalen auf den Adress- und Datenbusleitungen einerseits und den 4 Steuerleitungen $\overline{RD}$, $\overline{WR}$, $\overline{MREQ}$, $\overline{IORQ}$ andererseits. Das Lesesignal $\overline{RD}$ = 0 wird in den Maschinenzyklen M1 und M2 ausgegeben, wenn die Eingangsdaten für die CPU auf dem Datenbus anstehen; das Schreibsignal $\overline{WR}$ = 0 wird im Zyklus M3 ausgegeben, solange die Ausgangsdaten aus der CPU auf dem Datenbus anstehen. Gleichzeitig muß dementsprechend entweder die Speichereinheit (durch $\overline{MREQ}$ = 0) oder die Ein/Ausgabe-Einheit (durch $\overline{IORQ}$ = 0) aktiviert sein.

Anmerkungen:

1) Da die 8-Bit-Kanalnummer im 3. Maschinenzyklus nur die unteren 8 Leitungen A0 - A7 des Adressbusses belegt, werden beim Z80 zu diesem Zeitpunkt die höherwertigen 8 Adressleitungen A8 - A15 anderweitig benützt: es liegt entweder der Inhalt des Akkus A oder,bei indirekter Adressierung, der Inhalt des Registers B auf diesen Adressleitungen.

2) Bei Ein- und Ausgabezyklen (hier M3) wird beim Z80 automatisch zwischen dem 2. und 3. Takt ein Wartetakt T_W eingeschoben, um die Zugriffszeit zu verlängern, weil bei diesen Maschinenzyklen die Signale $\overline{RD}$/$\overline{WR}$ und $\overline{IORQ}$ einen halben Takt später erscheinen. Daher benötigt der Befehl OUT (n),A beim Z80 insgesamt 11 Takte gegenüber nur 10 Takten beim 8080.

3) Im Takt 3 und 4 des 1. Maschinenzyklus M1 legt der Mikroprozessor Z80 jedesmal die 7-Bit-Refresh-Adresse, die aus dem Refresh-Register R kommt und automatisch bei jedem Befehl um 1 erhöht wird, auf den Adressbus und ermöglicht so den Refresh-Vorgang für dynamische RAM-Bausteine. Gleichzeitig wird über die $\overline{RFSH}$-Leitung diesen Bausteinen durch Low-Signal mitgeteilt, daß sie den Refresh-Vorgang durchführen können. Da der Refresh-Vorgang immer im 1. Maschinenzyklus in Takt 3 und 4 abläuft, d.h. zeitlich immer dann, wenn die CPU gerade den Opcode dekodiert, kommt es zu keiner Behinderung der CPU bei der Bus-Benützung. (Die 7-Bit-Refreshadresse benötigt nur die Leitungen A0 - A6 des Adresbusses, daher wird A7 = 0 gesetzt, und auf den Leitungen A8 - A15 wird gleichzeitig der Inhalt des Interruptregisters I übertragen.)

12.4) Das Interrupt-System des Mikroprozessors Z80

Die Möglichkeiten für Programmunterbrechungen (Interrupts) wurden beim Mikroprozessor Z80 gegenüber dem 8080 erweitert. Der Mikroprozessor Z80 besitzt 2 Interrupt-Eingänge (Abb. 27): einen nicht-maskierbaren Eingang $\overline{NMI}$ (Non-Maskable Interrupt) und einen maskierbaren Eingang $\overline{INT}$. Der Interrupt-Eingang $\overline{INT}$ kann softwaremäßig gesperrt werden ("maskiert" werden) durch den Befehl DI (Disable Interrupt) und freigegeben werden durch den Befehl EI (Enable Interrupt). Der Mikroprozessor Z80 besitzt hierzu 2 Interrupt-Flipflops FF1 und FF2, wobei das Hauptflipflop FF1 uns vom Mikroprozessor 8080 her bekannt ist und der Sperrung bzw. der Freigabe des $\overline{INT}$-Eingangs dient. Das Hilfsflipflop FF2 wird als Zwischenspeicher benützt, solange ein $\overline{NMI}$-Unterbrechungsprogramm läuft (siehe unten).

Der Interrupt-Eingang $\overline{NMI}$ des Mikroprozessors Z80 hat eine höhere Priorität als der Eingang $\overline{INT}$ und dient, da er nicht maskierbar ist (d.h. softwaremäßig durch DI nicht gesperrt werden kann), zur Entgegennahme von Unterbrechungsanforderungen höchster Priorität, etwa zum Aufruf von Rettungs-Routinen bei Spannungsausfall usw. Der Eingang ist Low-aktiv, eine fallende Flanke löst den Interrupt-Mechanismus aus, der bewirkt, daß nach Beendigung des gerade in Arbeit befindlichen Befehls der Prozessor das Hauptprogramm unterbricht und zur festen Speicheradresse 0066H springt und die ab dort niedergelegte Unterbrechungs-Service-Routine ausführt. Diese Interrupt-Routine muß mit dem speziellen Rücksprungbefehl RETN (Return from Nonmaskable Interrupt) abgeschlossen sein, damit der Prozessor das Hauptprogramm an der unterbrochenen Stelle fortsetzen kann. (Beim Ablauf des NMI-Interrupt-Mechanismus wird gleich zu Beginn automatisch der zweite Interrupt-Eingang $\overline{INT}$ durch Rücksetzen des Flipflops FF1 gesperrt. Der augenblickliche Zustand des Flipflops FF1 wird jedoch zuvor ins Hilfsflipflop FF2 gerettet und nach Beendigung des Interrupt-Programms automatisch wieder ins Flipflop FF1 zurückgebracht, so daß der Anfangszustand wiederhergestellt wird.)

Der maskierbare Interrupt-Eingang $\overline{INT}$ des Mikroprozessors Z80 wird für 3 verschiedene Interrupt-Betriebsarten verwendet, die durch Programmbefehle eingestellt werden können:

Interrupt-Modus 0. Diese Betriebsart entspricht genau dem 8080-Interrupt-Modus. Sie wird durch den Befehl IM 0 (Interrupt Mode 0) bzw. automatisch durch ein RESET eingestellt. Wird von einem peripheren Gerät ein Interrupt mittels eines Low-Signals auf der $\overline{INT}$-Leitung angefordert und ist diese Leitung nicht maskiert (durch vorhergehendes DI), so werden die Ausgänge $\overline{M1}$ und $\overline{IORQ}$ vom Mikroprozessor Z80 gleichzeitig auf Null gelegt: dies entspricht dem Quittungs-Signal $\overline{INTA}$ des 8080. Das Interrupt-Quittungssignal "$\overline{INTA}$" wird also beim Z80 gebildet durch logisches ODERieren von $\overline{M1}$ und $\overline{IORQ}$! Dieses Interrupt-Quittungssignal "$\overline{INTA}$" bewirkt nun in bekannter Weise, daß das periphere anfordernde Gerät einen der 8 RESTART-Befehle auf den Datenbus legt, wodurch die CPU veranlaßt wird, eine Unterbrechungs-Service-Routine bei einer der 8 Restart-Speicheradressen 0000 bis 0038H aufzurufen. Durch den Return-Befehl RETI (oder RET) am Ende der Interrupt-Routine springt die CPU in das unterbrochene Programm zurück. (Zuvor muß durch den Befehl EI die Interruptmöglichkeit wieder freigegeben werden.)

Interrupt-Modus 1. Diese Betriebsart wird durch den Befehl IM 1 eingestellt. Wird in dieser Betriebsart über die $\overline{INT}$-Leitung eine Unterbrechung ausgelöst (wie oben), so startet der Mikroprozessor Z80 eine Interrupt-Service-Routine an der festen Speicheradresse 0038H (entspricht RESTART 7).

Der Interrupt-Modus 2 wird durch den Befehl IM 2 eingestellt. Diese Betriebsart ist am leistungsfähigsten, aber auch kompliziert. Sie ermöglicht den Aufruf einer Unterbrechungs-Routine an einer beliebigen Stelle im Speicher. Der Aufruf der Startadresse für die Interrupt-Routine erfolgt dabei indirekt über eine "Zeiger-Adresse". Das Verfahren arbeitet wie folgt:
Die 16-Bit-Startadressen sämtlicher vorkommender Interrupt-Routinen (maximal 128) werden vom Anwender in einer Tabelle

zusammengestellt und im Speicher an geeigneter Stelle als zusammenhängender Block abgespeichert (Startadressen-Block). Wenn nun ein peripheres Gerät die Ausführung einer bestimmten Interrupt-Routine wünscht, so legt es ein Low-Signal an den $\overline{INT}$-Eingang des Z80 und die CPU muß daraufhin das entsprechende Interrupt-Programm aufsuchen und starten. Dies geschieht durch die Bildung einer Zeiger-Adresse, welche den gewünschten Platz im Startadressen-Block aufruft und die dort befindliche Startadresse (die 2 aufeinanderfolgende Speicherplätze belegt) abruft und in den Programmzähler lädt, so daß die CPU als nächstes die angeforderte Interrupt-Routine ausführt.
Die Bildung der 16-Bit-Zeigeradresse erfolgt in Zusammenarbeit zwischen Mikroprozessor und Peripherie: Das höhere Byte der Zeigeradresse wird durch den Inhalt des Interrupt-Registers I gebildet, während das niedere Byte vom anfordernden peripheren Gerät über den Datenbus an die CPU gesandt wird und dort an das höhere Byte angehängt wird. Das Register I in der CPU muß also zuvor vom Anwender mit dem gewünschten Wert geladen worden sein *). (Der Aufruf einer Interrupt-Routine nach dem beschriebenen Modus 2 erfordert insgesamt 19 Takte!)
Für die Anwendung des Interrupt-Modus 2 sind insbesondere die speziellen Ein/Ausgabe-Bausteine PIO, SIO etc. des Systems Z80 vorgesehen, die eine Prioritäts-Rangfolge (daisy chain) für Mehrfach-Interrupts besitzen und die entsprechende Verarbeitung verschachtelter Interrupts erlauben. Auf diese speziellen Ein/Ausgabe-Bausteine werden wir später noch näher eingehen.
Alle Interrupt-Routinen müssen mit dem Befehl RETI (Return from Interrupt) abgeschlossen werden; bereits zuvor muß im Programm durch den Befehl EI die Interrupt-Möglichkeit wieder freigegeben worden sein.

---

*) Der gesamte Startadressen-Block muß im Speicher "auf 1 Seite" angeordnet werden, d.h. auf Speicherplätzen mit einheitlichem höherem Adress-Byte (z.B. auf den Plätzen 4E00 bis 4EFF).Das ergibt gerade 256:2 = 128 mögliche Startadressen im Startadressen-Block. Dieses höhere Adressbyte (4E) wird vom Anwender durch Programmbefehl im Register I niedergelegt.

13) DER BEFEHLSSATZ DES MIKROPROZESSORS Z80

Der Befehlssatz des Mikroprozessors Z80 umfaßt eine auf den ersten Blick verwirrende Vielzahl von Befehlen. Es soll daher in diesem Kapitel zunächst versucht werden, einen Überblick über die insgesamt 694 "autorisierten" Befehle des Z80 zu gewinnen, indem die Befehle in Befehlsklassen zusammengefaßt werden. Die mnemonische ASSEMBLER-Schreibweise der Befehle des Z80 erleichtert dabei sehr das Verständnis und den Überblick.
Soweit die mnemonische Schreibweise der Befehle nicht schon "selbst-erklärend" ist, wird sodann die Bedeutung der einzelnen Befehle in einer systematischen alphabetischen Zusammenstellung der Mnemonics erläutert, wobei die Kurzbeschreibung der Befehle auf englisch erfolgt, um den Zusammenhang mit der mnemonischen Schreibweise aufzuzeigen.
Eine Zusammenstellung aller Z80-Befehle in numerischer, hexadezimaler Reihenfolge wird dieses Kapitel abschließen.
Für weitergehende Information wird auf die Z80-ASSEMBLER-Literatur der Herstellerfirmen verwiesen (siehe Literaturverzeichnis am Schluß).

13.1) Einführung

Wie wir beim Mikroprozessor 8080 gesehen hatten, sind mit Operationscodes von 1 Byte Länge nur insgesamt $2^8 = 256$ verschiedene Befehle möglich. Der Mikroprozessor 8080 nützt hiervon 244 Opcodes aus. Es war das Bestreben bei der Entwicklung des Mikroprozessors Z80, mit dem Mikroprozessor 8080 aufwärts-kompatibel zu bleiben, d.h. der Befehlssatz des Z80 sollte die Befehle des 8080 als Untermenge enthalten. Bei der Erweiterung des Befehlssatzes des Z80 auf etwa 700 Befehle war man daher gezwungen, bei den neu hinzukommenden Befehlen größtenteils auf 2-Byte-Opcodes überzugehen. Der Befehlssatz des Mikroprozessors Z80 enthält also 1-Byte- und 2-Byte-Opcodes.

Befehle mit 1-Byte-Operationscode:

Die Z80-Befehle mit 1-Byte-Opcode umfassen im wesentlichen den Befehlssatz des 8080 mit seinen 244 Befehlen. Im Maschinencode sind die Befehle des 8080 mit den entsprechenden Befehlen des Z80 identisch, jedoch wurden in der ASSEMBLER-Schreibweise unterschiedliche symbolische Bezeichnungen (Mnemonics) gewählt.

So lautet beispielsweise der Maschinencode-Befehl 47(H)
in 8080-ASSEMBLER: MOV B,A
in Z80-ASSEMBLER: LD B,A .

Dieser Unterschied in der ASSEMBLER-Schreibweise des Z80 bedarf einer gewissen Gewöhnung; es wurden daher in einem Anhang auf Seite 196 die 8080-Befehle den entsprechenden äquivalenten Z80-Befehlen gegenübergestellt.Das Rückblättern auf diese uns schon vertrauten 244 Befehle des Mikroprozessors 8080 (Seite 86-105) und ein Vergleich mit den zugehörigen neuen Z80-Mnemonics wird als Einstieg in den Z80-Befehlssatz empfohlen.

Neben den 244 8080-Befehlen gibt es im Z80-Befehlssatz 8 neue Befehle mit 1-Byte-Opcode. Es sind dies die Befehle:

Hex-Code	Mnemonic	Bedeutung
08	EX AF,AF'	Vertausche Register A,F mit A',F'
10..	DJNZ d	relative Sprungbefehle zur Adresse PC+d (aktueller Programmzähler PC + Distanzzahl d)
18..	JR d	
20..	JR NZ,d	
28..	JR Z,d	
30..	JR NC,d	
38..	JR C,d	
D9	EXX	Vertausche Register B-L mit B'-L'

Die beiden Austauschbefehle EX AF,AF' und EXX greifen auf den zweiten Registersatz des Z80 zurück.

Befehle mit 2-Byte-Operationscode:

Die Befehle mit 2-Byte-Opcode stellen Erweiterungen gegenüber dem 8080-Befehlssatz dar. Sämtliche 2-Byte-Opcodes des Z80 beginnen entweder mit CB oder DD oder ED oder FD. Dieses 1. Befehlsbyte gibt schon eine gewisse Auskunft über die Art des Befehles:
Es beginnen mit

DD: Befehle unter Benützung des Indexregisters IX
FD: Befehle unter Benützung des Indexregisters IY
CB: Einzelbitbefehle und zusätzliche Rotations- und Schiebebefehle
ED: übrige Befehle mit 2-Byte-Opcode.

Da ein Befehl, der einen 2-Byte-Opcode besitzt, auch noch einen 2-Byte-Operanden besitzen kann, kommen im Z80-Befehlssatz Befehle mit 4 Byte Gesamtlänge vor!

Angabe von Speicherplätzen bei den Z80-Befehlen:

Der Aufruf des Inhalts von Speicherplätzen (und Kanalnummern) erfolgt in der Z80-ASSEMBLER-Schreibweise konsequent durch Angabe des Speicherplatzes in runden Klammern.
Die Angabe des Speicherplatzes kann dabei auf 3 Arten erfolgen:

- direkte (absolute) Angabe des Speicherplatzes durch eine 16-Bit-Adresse: (nn)
- indirekte Angabe des Speicherplatzes durch ein 16-Bit-Registerpaar rr, in welchem die Adresse steht: (rr)
- indizierte Angabe des Speicherplatzes durch eines der beiden 16-Bit-Indexregister IX bzw. IY plus einer 8-Bit-Distanzzahl d: (IX + d) bzw. (IY + d). Die wirksame Adresse ergibt sich durch Addition der Distanzzahl d zum Inhalt des betreffenden Indexregisters. Der Wertebereich der Distanzzahl d erstreckt sich von -128(D) bis +127(D).

Beispiele für Speicherplatzangaben:

direkt: LD A,(012F) : Lade Akku mit Inhalt des Speicherplatzes 012F.

indirekt: LD A,(BC) : Lade Akku mit Inhalt desjenigen Speicherplatzes, dessen Adresse im Registerpaar BC steht.

indiziert: LD A,(IX+5E): Lade Akku mit Inhalt desjenigen Speicherplatzes, dessen Adresse durch das Indexregister IX plus der Distanzzahl 5E gegeben ist. (Ist beispielsweise IX:=0134, so lautet die wirksame Adresse 0134+5E=0192).

13.2) Zusammenfassung der Z80-Befehle in Befehlsklassen

In den folgenden Tabellen sind die 694 Befehle des Mikroprozessors Z80 übersichtlich, nach Befehlsklassen geordnet, zusammengestellt. Diese Befehlsklassen werden anschließend im einzelnen erläutert. Die Bedeutung der Befehle wird mit unseren Vorkenntnissen meist schon unmittelbar aus der symbolischen ASSEMBLER-Schreibweise ersichtlich, eine systematische englische Kurzbeschreibung sämtlicher vorkommenden Z80-ASSEMBLER-Befehle wird im nächsten Abschnitt 13.3 gegeben werden. Diejenigen Befehle, die in den folgenden 3 Tabellen mit dem Zeichen + versehen sind, haben Einfluß auf den Zustand der Bedingungsbits (Flags); eine Zusammenstellung sämtlicher Befehle, welche die Flags beeinflussen, sowie die Art der Flag-Beeinflussung befindet sich im Abschnitt 13.4 (S.192).

Anmerkung: Das Bestreben, mit dem 8080-Befehlssatz kompatibel zu bleiben, führte beim Z80 dazu, daß einzelne Befehle im Z80-Befehlssatz doppelt auftreten. Zum Beispiel haben die beiden Befehle RLA (Maschinencode 17) und RL A (Maschinencode CB17) die gleiche Funktion!

8-Bit-Transferbefehle

	n	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	(BC)	(DE)	(nn)	I	R
LD A,-	3Exx	7F	78	79	7A	7B	7C	7D	7E	DD7Exx	FD7Exx	0A	1A	3Axxxx	ED57	ED5F
LD B,-	06xx	47	40	41	42	43	44	45	46	DD46xx	FD46xx					
LD C,-	0Exx	4F	48	49	4A	4B	4C	4D	4E	DD4Exx	FD4Exx					
LD D,-	16xx	57	50	51	52	53	54	55	56	DD56xx	FD56xx					
LD E,-	1Exx	5F	58	59	5A	5B	5C	5D	5E	DD5Exx	FD5Exx					
LD H,-	26xx	67	60	61	62	63	64	65	66	DD66xx	FD66xx					
LD L,-	2Exx	6F	68	69	6A	6B	6C	6D	6E	DD6Exx	FD6Exx					
LD (HL),-	36xx	77	70	71	72	73	74	75								
LD (IX+d),-	DD36xxxx	DD77xx	DD70xx	DD71xx	DD72xx	DD73xx	DD74xx	DD75xx								
LD (IY+d),-	FD36xxxx	FD77xx	FD70xx	FD71xx	FD72xx	FD73xx	FD74xx	FD75xx								
LD (BC),-		02														
LD (DE),-		12														
LD (nn),-		32xxxx														
LD I,-		ED47														
LD R,-		ED4F														

16-Bit-Transferbefehle

	BC	DE	HL	SP	IX	IY
LD -,nn	01xxxx	11xxxx	21xxxx	31xxxx	DD21xxxx	FD21xxxx
LD -,(nn)	ED4Bxxxx	ED5Bxxxx	2Axxxx	ED7Bxxxx	DD2Axxxx	FD2Axxxx
LD (nn),-	ED43xxxx	ED53xxxx	22xxxx	ED73xxxx	DD22xxxx	FD22xxxx
LD SP,-			F9		DDF9	FDF9

	BC	DE	HL	AF	IX	IY
PUSH -	C5	D5	E5	F5	DDE5	FDE5
POP -	C1	D1	E1	F1	DDE1	FDE1

EX (SP),HL	E3
EX (SP),IX	DDE3
EX (SP),IY	FDE3
EX DE,HL	EB
EX AF,AF'	08
EXX	D9

Blocktransfer-/Datensuch-Befehle

* LDI	EDA0	* CPI	EDA1
* LDIR	EDB0	* CPIR	EDB1
* LDD	EDA8	* CPD	EDA9
* LDDR	EDB8	* CPDR	EDB9

Eingabe- und Ausgabe-Befehle

IN A,(n) DBxx
OUT (n),A D3xx

	A	B	C	D	E	H	L
* IN -,(C)	ED78	ED40	ED48	ED50	ED58	ED60	ED68
OUT (C),-	ED79	ED41	ED49	ED51	ED59	ED61	ED69

Block-Eingabe/Ausgabe

* INI	EDA2	* OUTI	EDA3
* INIR	EDB2	* OTIR	EDB3
* IND	EDAA	* OUTD	EDAB
* INDR	EDBA	* OTDR	EDBB

8-Bit-Arithmetische und Logische Befehle

	n	B	C	D	E	H	L	(HL)	A	(IX+d)	(IY+d)
+ ADD A,-	C6xx	80	81	82	83	84	85	86	87	DD86xx	FD86xx
+ ADC A,-	CExx	88	89	8A	8B	8C	8D	8E	8F	DD8Exx	FD8Exx
+ SUB -	D6xx	90	91	92	93	94	95	96	97	DD96xx	FD96xx
+ SBC A,-	DExx	98	99	9A	9B	9C	9D	9E	9F	DD9Exx	FD9Exx
+ AND -	E6xx	A0	A1	A2	A3	A4	A5	A6	A7	DDA6xx	FDA6xx
+ XOR -	EExx	A8	A9	AA	AB	AC	AD	AE	AF	DDAExx	FDAExx
+ OR -	F6xx	B0	B1	B2	B3	B4	B5	B6	B7	DDB6xx	FDB6xx
+ CP -	FExx	B8	B9	BA	BB	BC	BD	BE	BF	DDBExx	FDBExx
+ INC -		04	0C	14	1C	24	2C	34	3C	DD34xx	FD34xx
+ DEC -		05	0D	15	1D	25	2D	35	3D	DD35xx	FD35xx

+ DAA 27
+ CPL 2F
+ NEG ED44

16-Bit-Arithmetische Befehle

	BC	DE	HL	SP	IX	IY
INC -	03	13	23	33	DD23	FD23
DEC -	0B	1B	2B	3B	DD2B	FD2B
+ ADD HL,-	09	19	29	39		
+ ADC HL,-	ED4A	ED5A	ED6A	ED7A		
+ SBC HL,-	ED42	ED52	ED62	ED72		
+ ADD IX,-	DD09	DD19		DD39	DD29	
+ ADD IY,-	FD09	FD19		FD39		FD29

Rotations- und Schiebe-Befehle

	B	C	D	E	H	L	(HL)	A	(IX+d)	(IY+d)
+ RLC -	CB00	CB01	CB02	CB03	CB04	CB05	CB06	CB07	DDCBxx06	FDCBxx06
+ RRC -	CB08	CB09	CB0A	CB0B	CB0C	CB0D	CB0E	CB0F	DDCBxx0E	FDCBxx0E
+ RL -	CB10	CB11	CB12	CB13	CB14	CB15	CB16	CB17	DDCBxx16	FDCBxx16
+ RR -	CB18	CB19	CB1A	CB1B	CB1C	CB1D	CB1E	CB1F	DDCBxx1E	FDCBxx1E
+ SLA -	CB20	CB21	CB22	CB23	CB24	CB25	CB26	CB27	DDCBxx26	FDCBxx26
+ SRA -	CB28	CB29	CB2A	CB2B	CB2C	CB2D	CB2E	CB2F	DDCBxx2E	FDCBxx2E
+ SRL -	CB38	CB39	CB3A	CB3B	CB3C	CB3D	CB3E	CB3F	DDCBxx3E	FDCBxx3E

+ RLCA 07
+ RRCA 0F
+ RLA 17
+ RRA 1F
+ RLD ED6F
+ RRD ED67

Sprungbefehle

JP nn	C3xxxx	CALL nn	CDxxxx			RET	C9
JP (HL)	E9					RETI	ED4D
JP (IX)	DDE9					RETN	ED45
JP (IY)	FDE9						
JR d	18xx						

	Z	NZ	C	NC	PE	PO	M	P
JP -,nn	CAxxxx	C2xxxx	DAxxxx	D2xxxx	EAxxxx	E2xxxx	FAxxxx	F2xxxx
CALL -,nn	CCxxxx	C4xxxx	DCxxxx	D4xxxx	ECxxxx	E4xxxx	FCxxxx	F4xxxx
RET -	C8	C0	D8	D0	E8	E0	F8	F0
JR -,d	28xx	20xx	38xx	30xx				

DJNZ d 10xx

RST -	0	8	10	18	20	28	30	38
	C7	CF	D7	DF	E7	EF	F7	FF

Einzelbit-Befehle

	B	C	D	E	H	L	(HL)	A	(IX+d)	(IY+d)
+ BIT 0,-	CB40	CB41	CB42	CB43	CB44	CB45	CB46	CB47	DDCBxx46	FDCBxx46
+ BIT 1,-	CB48	CB49	CB4A	CB4B	CB4C	CB4D	CB4E	CB4F	DDCBxx4E	FDCBxx4E
+ BIT 2,-	CB50	CB51	CB52	CB53	CB54	CB55	CB56	CB57	DDCBxx56	FDCBxx56
+ BIT 3,-	CB58	CB59	CB5A	CB5B	CB5C	CB5D	CB5E	CB5F	DDCBxx5E	FDCBxx5E
+ BIT 4,-	CB60	CB61	CB62	CB63	CB64	CB65	CB66	CB67	DDCBxx66	FDCBxx66
+ BIT 5,-	CB68	CB69	CB6A	CB6B	CB6C	CB6D	CB6E	CB6F	DDCBxx6E	FDCBxx6E
+ BIT 6,-	CB70	CB71	CB72	CB73	CB74	CB75	CB76	CB77	DDCBxx76	FDCBxx76
+ BIT 7,-	CB78	CB79	CB7A	CB7B	CB7C	CB7D	CB7E	CB7F	DDCBxx7E	FDCBxx7E
RES 0,-	CB80	CB81	CB82	CB83	CB84	CB85	CB86	CB87	DDCBxx86	FDCBxx86
RES 1,-	CB88	CB89	CB8A	CB8B	CB8C	CB8D	CB8E	CB8F	DDCBxx8E	FDCBxx8E
RES 2,-	CB90	CB91	CB92	CB93	CB94	CB95	CB96	CB97	DDCBxx96	FDCBxx96
RES 3,-	CB98	CB99	CB9A	CB9B	CB9C	CB9D	CB9E	CB9F	DDCBxx9E	FDCBxx9E
RES 4,-	CBA0	CBA1	CBA2	CBA3	CBA4	CBA5	CBA6	CBA7	DDCBxxA6	FDCBxxA6
RES 5,-	CBA8	CBA9	CBAA	CBAB	CBAC	CBAD	CBAE	CBAF	DDCBxxAE	FDCBxxAE
RES 6,-	CBB0	CBB1	CBB2	CBB3	CBB4	CBB5	CBB6	CBB7	DDCBxxB6	FDCBxxB6
RES 7,-	CBB8	CBB9	CBBA	CBBB	CBBC	CBBD	CBBE	CBBF	DDCBxxBE	FDCBxxBE
SET 0,-	CBC0	CBC1	CBC2	CBC3	CBC4	CBC5	CBC6	CBC7	DDCBxxC6	FDCBxxC6
SET 1,-	CBC8	CBC9	CBCA	CBCB	CBCC	CBCD	CBCE	CBCF	DDCBxxCE	FDCBxxCE
SET 2,-	CBD0	CBD1	CBD2	CBD3	CBD4	CBD5	CBD6	CBD7	DDCBxxD6	FDCBxxD6
SET 3,-	CBD8	CBD9	CBDA	CBDB	CBDC	CBDD	CBDE	CBDF	DDCBxxDE	FDCBxxDE
SET 4,-	CBE0	CBE1	CBE2	CBE3	CBE4	CBE5	CBE6	CBE7	DDCBxxE6	FDCBxxE6
SET 5,-	CBE8	CBE9	CBEA	CBEB	CBEC	CBED	CBEE	CBEF	DDCBxxEE	FDCBxxEE
SET 6,-	CBF0	CBF1	CBF2	CBF3	CBF4	CBF5	CBF6	CBF7	DDCBxxF6	FDCBxxF6
SET 7,-	CBF8	CBF9	CBFA	CBFB	CBFC	CBFD	CBFE	CBFF	DDCBxxFE	FDCBxxFE

CPU-Steuerbefehle

							IM 0	ED46
NOP	00	+ SCF	37		EI	FB	IM 1	ED56
HALT	76	+ CCF	3F		DI	F3	IM 2	ED5E

Erläuterungen zu den einzelnen Befehlsklassen

8- und 16-Bit-Transferbefehle

Die 8- und 16-Bit-Transferbefehle des Z80 werden einheitlich durch die mnemonische Bezeichnung LD (Load) gekennzeichnet, unter Angabe von Ziel und Quelle des Transfers:

```
LD Ziel,Quelle
```

Bei den 8-Bit-Transferbefehlen des Z80 handelt es sich überwiegend um die alten 8080-Transferbefehle. Neu hinzugekommen sind die Befehle mit indizierter Angabe eines Speicherplatzes (IX+d) bzw. (IY+d), sowie Transferbefehle zwischen den neuen Registern I bzw. R und dem Akku A.

Die Zahl der 16-Bit-Transferbefehle wurde stark erweitert. Sie umfassen jetzt auch den 16-Bit-Datenaustausch zwischen dem Speicher und den universellen Registerpaaren der CPU sowie den Indexregistern IX und IY und dem Stackpointer SP.

Bei den Austauschbefehlen (Exchange) sind die 16-Bit-Plätze angegeben, zwischen denen der Datenaustausch erfolgen soll. Lediglich beim Befehl EXX sind die auszutauschenden Register nicht explizit angegeben; hierbei wird gleichzeitig vertauscht: BC mit B'C', DE mit D'E' und HL mit H'L'.
Hinweis: Nur mit den beiden Befehlen EXX und EX AF,AF' kann auf den 2. Registersatz A',F',B'-L' zurückgegriffen werden!

Die Push- und Pop-Befehle (zur vorübergehenden Rettung von Registerinhalten in den Stack) wurden auf die Indexregister IX und IY ausgedehnt.

Blocktransfer- und Datensuch-Befehle

Blocktransferbefehle sind Befehle, die nicht nur 1 oder 2 Bytes übertragen, sondern die einen ganzen Block von Daten innerhalb des Speichers verschieben können. Diese Klasse von Befehlen ist neu und sehr nützlich. Blocktransfer-Operationen mußten beim Mikroprozessor 8080 noch aus mehreren Einzelbefehlen aufgebaut werden. Sie sind beim Z80 durch einen einzigen Befehl durchführbar, der allerdings verständlicherweise viel Zeit zu seiner Ausführung benötigt.
Folgende 4 Blocktransferbefehle sind im Z80-Befehlssatz enthalten:

LDI: "Laden und inkrementieren". Lade den Speicherplatz (DE) mit dem Speicherplatz (HL), inkrementiere die Registerpaare DE und HL, dekrementiere das (Zähl-)Registerpaar BC.

LDIR: "Laden, inkrementieren und repetieren". Wie LDI, aber wiederholen bis BC=0 ist.

LDD: "Laden und dekrementieren". Lade den Speicherplatz (DE) mit dem Speicherplatz (HL), dekrementiere DE und HL, dekrementiere das (Zähl-)Registerpaar BC.

LDDR: "Laden, dekrementieren und repetieren". Wie LDD, aber wiederholen bis BC=0 ist.

Datensuchbefehle sind Befehle zum Aufsuchen von 8-Bit-Daten ("Zeichen") im Speicher. Das Suchen erfolgt durch Vergleich der Speicherplätze mit dem Akku. Ein "gefundenes Zeichen" wird durch das Zero-Flag Z=1 angezeigt.
Folgende 4 Suchbefehle kommen vor:

CPI: "Vergleichen und inkrementieren". Vergleiche (compare) den Speicherplatz (HL) mit dem Akku, inkrementiere HL, dekrementiere das (Zähl-)Registerpaar BC.

CPIR: "Vergleichen, inkrementieren und repetieren". Wie CPI, aber wiederholen bis das Zeichen gefunden wurde oder BC=0 ist.

CPD: "Vergleichen und dekrementieren". Vergleiche den Speicherplatz (HL) mit dem Akku, dekrementiere HL, dekrementiere das (Zähl-)Registerpaar BC.

CPDR: "Vergleichen, dekrementieren und repetieren". Wie CPD, aber wiederholen bis das Zeichen gefunden wurde oder BC=0 ist.

Eingabe- und Ausgabe-Befehle

Die Eingabe und Ausgabe von Daten erfolgt (bei isolierter Ein/Ausgabe) durch die Befehle IN und OUT unter Angabe der Kanalnummer und des Registers, über das der Datentransfer laufen soll. Beim Mikroprozessor 8080 konnte nur der Akku mit der Peripherie Daten austauschen. Beim Mikroprozessor Z80 können darüber hinaus sämtliche universellen Register (A-L) mit der Peripherie in Datenaustausch treten, die Angabe der Kanalnummer erfolgt dabei indirekt durch das Register C.

Entsprechend den Blocktransfer-Möglichkeiten innerhalb des Speichers gibt es auch bei der Ein/Ausgabe die Möglichkeit, ganze Datenblöcke ein- oder auszugeben. Die Angabe der Kanalnummer erfolgt dabei indirekt durch das Register C.

Block-Eingabebefehle:

INI: "Eingeben und inkrementieren". Lade den Speicherplatz (HL) mit den Eingangsdaten von Kanal (C), inkrementiere HL, dekrementiere das (Zähl-)Register B.

INIR: "Eingeben, inkrementieren und repetieren". Wie INI, aber wiederholen bis B=0 ist.

IND: "Eingeben und dekrementieren". Lade den Speicherplatz (HL) mit den Eingangsdaten von Kanal (C), dekrementiere HL, dekrementiere das (Zähl-)Register B.

INDR: "Eingeben, dekrementieren und repetieren". Wie IND, aber wiederholen bis B=0 ist.

Block-Ausgabebefehle:

OUTI: "Ausgeben und inkrementieren". Gib den Inhalt des Speicherplatzes (HL) über den Kanal (C) aus, inkrementiere HL, dekrementiere das (Zähl-)Register B.

OTIR: "Ausgeben, inkrementieren und repetieren". Wie OUTI, aber wiederholen bis B=0 ist.

OUTD: "Ausgeben und dekrementieren". Gib den Inhalt des Speicherplatzes (HL) über den Kanal (C) aus, dekrementiere HL, dekrementiere das (Zähl-)Register B.

OTDR: "Ausgeben, dekrementieren und repetieren". Wie OUTD, aber wiederholen bis B=0 ist.

8-Bit-Arithmetische und -Logische Befehle

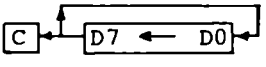
Bei den Z80-Befehlen für arithmetische und logische 8-Bit-Operationen handelt es sich im wesentlichen um die alten 8080-Befehle. Neu hinzugekommen sind die Befehle mit indizierter Angabe eines Speicherplatzes (IX+d) bzw. (IY+d), sowie der Negationsbefehl NEG, bei dem der Akku-Inhalt durch sein Zweierkomplement ersetzt wird.

16-Bit-Arithmetische Befehle

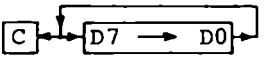
Die Zahl der 16-Bit-Arithmetikbefehle wurde beim Z80 stark erweitert. Neben Inkrementier- und Dekrementierbefehlen und den Addierbefehlen ADD, die jetzt auch auf die beiden Indexregister IX und IY anwendbar sind, wurden Additions- und Subtraktionsbefehle mit Berücksichtigung des Carry-Flags, ADC und SBC, in den Befehlssatz aufgenommen. Das Ergebnis der Additions- und Subtraktionsoperationen steht dabei in demjenigen Registerpaar, das im mnemonischen Befehl an erster Stelle genannt ist.

Rotations- und Schiebe-Befehle

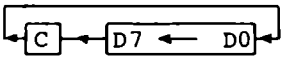
Diese Klasse von Befehlen ist praktisch neu im Z80-Befehlssatz. (Der Mikroprozessor 8080 kannte nur 4 1-Byte-Rotationsbefehle, die wegen der Kompatibilität beim Z80 beibehalten wurden: RLCA, RRCA, RLA, RRA.) Die neuen Rotations- und Schiebebefehle beziehen sich auf sämtliche universellen Register und auf Speicherplätze, die indirekt (über HL) oder indiziert (über IX+d bzw. IY+d) aufgerufen werden.
Es gibt 7 verschiedene Arten von Rotations- und Schiebebefehlen:



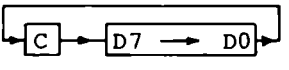
Rotiere Links Circular: RLC



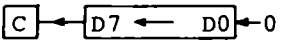
Rotiere Rechts Circular: RRC



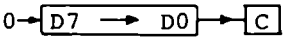
Rotiere Links (durch das Carrybit): RL



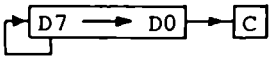
Rotiere Rechts (durch das Carrybit): RR



Schiebe Links Arithmetisch: SLA



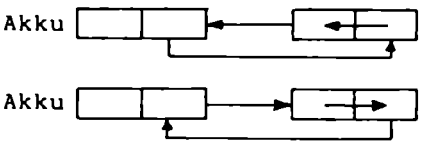
Schiebe Rechts Logisch: SRL



Schiebe Rechts Arithmetisch: SRA

Beim Rotieren unterscheidet man zwischen "zirkularem Rotieren" und "Rotieren durch das Carrybit hindurch", das Bitmuster rotiert jeweils um 1 Stelle weiter.
Beim Schieben wird der Inhalt des Registers oder Speicherplatzes um 1 Stelle nach links oder rechts geschoben, das hinausgeschobene Bit kommt ins Carrybit, das neu hereinkommende Bit ist 0. (Dies entspricht einer Multiplikation mit 2 bzw. einer Division durch 2.) Es gibt aber eine Ausnahme: Arithmetisches Rechts-Schieben SRA. Da bei arithmetischen Operationen das linke Bit das Vorzeichen angibt (Zweierkomplement-Arithmetik), soll beim arithmetischen Rechts-Schieben dieses Vorzeichen erhalten bleiben: das Vorzeichenbit D7 wird daher bei diesem Befehl reproduziert.

2 spezielle Rotationsbefehle für Halb-Bytes (BCD-Ziffern: Digit) sind beim Z80 zusätzlich vorhanden, um das Rechnen mit BCD-Zahlen zu erleichtern. Bei diesen beiden Befehlen werden Halb-Bytes zwischen dem Akkumulator und einem Speicherplatz (durch HL indirekt adressiert) auf folgende Weise ausgetauscht:



Rotiere Links Digit:RLD

Rotiere Rechts Digit:RRD

Sprungbefehle:

Die Sprungbefehle im Z80-Befehlssatz sind weitgehend die alten 8080-Befehle. Neu hinzugekommen sind vor allem (bedingte und unbedingte) relative Sprungbefehle JR unter Angabe einer Distanzzahl d, die von -128 bis +127 laufen kann. Diese Distanzzahl d wird zum aktuellen Programmzähler PC hinzuaddiert und ergibt den neuen Programmzählerstand PC:=PC+d ; dieser Befehl wird von der CPU als nächster ausgeführt.

DJNZ d: "Dekrementiere B und springe relativ nach PC+d, solange B≠0 ". Dieser relative Sprungbefehl wird vorteilhaft bei Programmschleifen eingesetzt. Der Sprung zur Adresse PC+d erfolgt immer, solange das Zählregister B noch nicht bis auf 0 dekrementiert worden ist.

Der Mikroprozessor Z80 besitzt 3 Return-Befehle:

RET: Rücksprung vom Unterprogramm (der alte 8080-Returnbefehl)
RETI: Rücksprung vom Interrupt-Programm
RETN: Rücksprung vom nichtmaskierbaren Interrupt-Programm

Einzelbit-Befehle:

Die 240 Einzelbitbefehle des Z80-Befehlssatzes erlauben das Setzen (SET), Rücksetzen (RES), sowie Testen (BIT) einzelner Bits in sämtlichen universellen Registern (A-L) und in allen Speicherplätzen, die indirekt (über HL) oder indiziert (über IX+d bzw. IY+d) aufgerufen werden. Das Bit-Testen erfolgt in der Weise, daß vom Prozessor das Zero-Flag Z=1 gesetzt wird, wenn das getestete Bit 0 ist.

CPU-Steuerbefehle

Zu den 6 alten 8080-Steuerbefehlen kommen 3 Befehle (IM 0, IM 1, IM 2) hinzu, welche den gewünschten Interrupt-Modus des Mikroprozessors Z80 einzustellen erlauben.

13.3) Alphabetische Zusammenstellung der Z80-Mnemonics

Auf den folgenden Seiten sind die mnemonischen Bezeichnungen der Z80-Befehle in alphabetischer Reihenfolge zusammengestellt. Eine englische Kurzbeschreibung der Befehlsinhalte soll die Bedeutung der einzelnen Befehle erläutern und den Bezug zu den mnemonischen Abkürzungen herstellen.
Für jeden Befehl ist die von der CPU zu seiner Durchführung benötigte Zahl von Maschinenzyklen und Takten angegeben.

MNEMONICS DES Z80

****

```
                                                                  Zyklen/Takte
ADC A,n          Add with carry value n    to Acc.  ............. 2/7
ADC A,r          Add with carry Reg.r      to Acc.  ............. 1/4
ADC A,(HL)       Add with carry loc.(HL)   to Acc.  ............. 2/7
ADC A,(IX,d)     Add with carry loc.(IX+d) to Acc.  ............. 5/19
ADC A,(IY,d)     Add with carry loc.(IY+d) to Acc.  ............. 5/19
ADC HL,rr        Add with carry Reg.pair rr to HL  .............. 4/15
ADD A,n          Add value n    to Acc.  ........................ 2/7
ADD A,r          Add Reg.r      to Acc.  ........................ 1/4
ADD A,(HL)       Add loc.(HL)   to Acc.  ........................ 2/7
ADD A,(IX+d)     Add loc.(IX+d) to Acc.  ........................ 5/19
ADD A,(IY+d)     Add loc.(IY+d) to Acc.  ........................ 5/19
ADD HL,rr        Add Reg.pair rr to HL  ......................... 3/11
ADD IX,rr        Add Reg.pair rr to IX  ......................... 4/15
ADD IY,rr        Add reg.pair rr to IY  ......................... 4/15
AND n            'AND' value n     and Acc.  .................... 2/7
AND r            'AND' Reg.r       and Acc.  .................... 1/4
AND (HL)         'AND' loc.(HL)    and Acc.  .................... 2/7
AND (IX+d)       'AND' loc.(IX+d)  and Acc.  .................... 5/19
AND (IY+d)       'And' loc.(IY+d)  and Acc.  .................... 5/19
BIT b,r          Test Bit b of Reg.r        ..................... 2/8
BIT b,(HL)       Test Bit b of loc. (HL)    ..................... 3/12
BIT b,(IX+d)     Test Bit b of loc.(IX+d)   ..................... 5/20
BIT b,(IY+d)     Test Bit b of loc.(IY+d)   ..................... 5/20
CALL nn          Unconditional call subroutine at address nn  ... 5/17
CALL co,nn       Call subroutine at address nn if
                   condition co (Z,NZ,C,NC,PE,PO,M,P) is true ... 5/17
                                                                   (3/10)
CCF              Complement carry flag  ......................... 1/4
CP n             Compare value n     and Acc.  ................. 2/7
CP r             Compare Reg.r       and Acc.  ................. 1/4
CP (HL)          Compare loc.(HL)    and Acc.  ................. 2/7
CP (IX+d)        Compare loc.(IX+d)  and Acc.  ................. 5/19
CP (IY+d)        Compare loc.(IY+d)  and Acc.  ................. 5/19
CPD              Compare loc.(HL) and Acc.,decrement HL and BC .. 4/16
CPDR             Compare loc.(HL) and Acc.,
                   decrement HL and BC,           5/21 :BC#0 und  A#(HL)
                   repeat until BC=0  ........... 4/16 :BC=0 oder A=(HL)
CPI              Compare loc.(HL) and Acc.,
                   increment HL and decrement BC  ............... 4/16
CPIR             Compare loc.(HL) and Acc.,
                   increment HL,decrement BC,     5/21 :BC#0 und  A#(HL)
                   repeat until BC=0  ........... 4/16 :BC=0 oder A=(HL)
CPL              Complement Acc.  ............................... 1/4
DAA              Decimal adjust Acc.  ........................... 1/4
DEC r            Decrement Reg.r  ............................... 1/4
DEC (HL)         Decrement loc.(HL)  ............................ 3/11
DEC (IX+d)       Decrement loc.(IX+d)  .......................... 6/23
DEC (IY+d)       Decrement loc.(IY+d)  .......................... 6/23
```

```
DEC rr          Decrement Reg.pair rr  ..........................  1/6
DEC IX          Decrement IX  ..................................  2/10
DEC IY          Decrement IY  ..................................  2/10
DJNZ d          Decrement B and jump relative to PC+d ,if B#0 ..  3/13
                                                                   (2/8)
DI              Disable interrupts  ............................  1/4
EI              Enable interrupts  .............................  1/4
EX (SP),HL      Exchange stack-loc.(SP)/(SP+1) with HL  ........  5/19
EX (SP),IX      Exchange stack-loc.(SP)/(SP+1) with IX  ........  6/23
EX (SP),IY      Exchange stack-loc.(SP)/(SP+1) with IY  ........  6/23
EX AF,AF'       Exchange the contents of AF and A'F'  ..........  1/4
EX DE,HL        Exchange the contents of DE and HL  ............  1/4
EXX             Exchange the contents of BC,DE,HL with
                  the contents of B'C',D'E',H'L' respectively ..  1/4
HALT            HALT (wait for interrupt or reset)  ............  1/4
IM 0            Set interrupt mode 0  ..........................  2/8
IM 1            Set interrupt mode 1  ..........................  2/8
IM 2            Set interrupt mode 2  ..........................  2/8
IN A,(n)        Load Acc.  with input from port n  .............  3/11
IN r,(C)        Load Reg.r with input from port (C)  ...........  3/12
IND             Load loc.(HL)  with input from port (C),
                  decrement HL and B  ..........................  4/16
INDR            Load loc.(HL) with input from port (C),
                  decrement HL and B,                      5/21  :B#0
                  repeat until B=0  ........................ 4/16  :B=0
INI             Load loc.(HL) with input from port (C),
                  increment HL and decrement B  ................  4/16
INIR            Load loc.(HL) with input from port (C),
                  increment HL and decrement B,                5/21  :B#0
                  repeat until B=0  ........................ 4/16  :B=0
INC r           Increment Reg.r  ...............................  1/4
INC (HL)        Increment loc.(HL)  ............................  3/11
INC (IX+d)      Increment loc.(IX+d)  ..........................  6/23
INC (IY+d)      Increment loc.(IY+d)  ..........................  6/23
INC rr          Increment Reg.pair rr  .........................  1/6
INC IX          Increment IX  ..................................  2/10
INC IY          Increment IY  ..................................  2/10
JP (HL)         Unconditional jump to address (HL)  ............  1/4
JP (IX)         Unconditional jump to address (IX)  ............  2/8
JP (IY)         Unconditional jump to address (IY)  ............  2/8
JP nn           Unconditional jump to address nn  ..............  3/10
JP co,nn        Jump to address nn ,if
                  condition co (Z,NZ,C,NC,PE,PO,M,P) is true ...  3/10
JR d            Unconditional jump relative to address PC+d  ...  3/12
JR co,d         Jump relative to address PC+d ,if
                  condition co (Z,NZ,C,NC) is true  ............  3/12
                                                                   (2/7)
LD A,(BC)       Load Acc. with loc.(BC)  .......................  2/7
LD A,(DE)       Load Acc. with loc.(DE)  .......................  2/7
LD A,(nn)       Load Acc. with loc.(nn)  .......................  4/13
LD A,I          Load Acc. with I ...............................  2/9
LD A,R          Load Acc. with R  ..............................  2/9
LD (BC),A       Load loc.(BC) with Acc.  .......................  2/7
LD (DE),A       Load loc.(DE) with Acc.  .......................  2/7
```

```
LD (HL),n       Load loc.(HL) with value n ..................... 3/10
LD (HL),r       Load loc.(HL) with Reg.r ....................... 2/7
LD HL,(nn)      Load HL with loc.(nn)/(nn+1) ................... 5/16
LD I,A          Load I with Acc. ............................... 2/9
LD IX,nn        Load IX with value nn .......................... 4/14
LD IX,(nn)      Load IX with loc.(nn)/(nn+1) ................... 6/20
LD (IX+d),n     Load loc.(IX+d) with value n ................... 5/19
LD (IX+d),r     Load loc.(IX+d) with Reg.r ..................... 5/19
LD IY,nn        Load IY with value nn .......................... 4/14
LD IY,(nn)      Load IY with loc.(nn)/(nn+1) ................... 6/20
LD (IY+d),n     Load loc.(IY+d) with value n ................... 5/19
LD (IY+d),r     Load loc.(IY+d) with Reg.r ..................... 5/19
LD (nn),A       Load loc.(nn) with Acc. ........................ 4/13
LD (nn),rr      Load loc.(nn)/(nn+1) with Reg.pair rr ......... 6/20
LD (nn),HL      Load loc.(nn)/(nn+1) with HL ................... 5/16
LD (nn),IX      Load loc.(nn)/(nn+1) with IX ................... 6/20
LD (nn),IY      Load loc.(nn)/(nn+1) with IY ................... 6/20
LD R,A          Load R with Acc. ............................... 2/9
LD r,n          Load Reg.r with value n ........................ 2/7
LD r,r          Load Reg.r with Reg.r .......................... 1/4
LD r,(HL)       Load Reg.r with loc.(HL) ....................... 2/7
LD r,(IX+d)     Load Reg.r with loc.(IX+d) ..................... 5/19
LD r,(IY+d)     Load Reg.r with loc.(IY+d) ..................... 5/19
LD rr,nn        Load Reg.pair rr with value nn ................. 3/10
LD rr,(nn)      Load Reg.pair rr with loc.(nn)/(nn+1) ......... 6/20
LD SP,HL        Load SP with HL ................................ 1/6
LD SP,IX        Load SP with IX ................................ 2/10
LD SP,IY        Load SP with IY ................................ 2/10
LDD             Load loc.(DE) with loc.(HL),
                  decrement DE,HL and BC ....................... 4/16
LDDR            Load loc.(DE) with loc.(HL),
                  decrement DE,HL and BC,                  5/21 :BC#0
                  repeat until BC=0 ........................ 4/16 :BC=0
LDI             Load loc.(DE) with loc.(HL),
                  increment DE,HL ,decrement BC ................ 4/16
LDIR            Load loc.(DE) with loc.(HL),
                  increment DE,HL ,decrement BC,           5/21 :BC#0
                  repeat until BC=0 ........................ 4/16 :BC=0
NEG             Negate Acc. (2's complement) ................... 2/8
NOP             No operation ................................... 1/4
OR n            'OR' value n    and Acc. ....................... 2/7
OR r            'OR' Reg.r      and Acc. ....................... 1/4
OR (HL)         'OR' loc.(HL)   and Acc. ....................... 2/7
OR (IX+d)       'OR' loc.(IX+d) and Acc. ....................... 5/19
OR (IY+d)       'OR' loc.(IY+d) and Acc. ....................... 5/19
OUT (C),r       Load output port (C) with Reg.r ................ 3/12
OUT (n),A       Load output port (n) with Acc. ................. 3/11
OUTD            Load output port (C) with loc.(HL),
                  decrement HL and B ........................... 4/16
OUTI            Load output port (C) with loc.(HL),
                  increment HL and decrement B ................. 4/16
OTDR            Load output port (C) with loc.(HL),
                  decrement HL and B,                      5/21 :B#0
                  repeat until B=0 ......................... 4/16 :B=0
```

OTIR	Load output port (C) with loc.(HL), increment HL ,decrement B,	5/21 :B#0
	repeat until B=0	4/16 :B=0
POP IX	Load IX with top of stack	4/14
POP IY	Load IY with top of stack	4/14
POP rr	Load Reg.pair rr with top of stack	3/10
PUSH IX	Load IX onto stack	4/15
PUSH IY	Load IY onto stack	4/15
PUSH rr	Load Reg.pair rr onto stack	3/11
RES b,r	Reset Bit b of Reg.r	4/8
RES b,(HL)	Reset Bit b of loc.(HL)	4/15
RES b,(IX+d)	Reset Bit b of loc.(IX+d)	6/23
RES b,(IY+d)	Reset Bit b of loc.(IY+d)	6/23
RET	Return from subroutine	3/10
RET co	Return from subroutine if condition co (Z,NZ,C,NC,PE,PO,M,P) is true ...	3/11 (1/5)
RETI	Return from interrupt	4/14
RETN	Return from non maskable interrupt	4/14
RLA	Rotate left Acc. thru carry	1/4
RL r	Rotate left Reg.r thru carry	2/8
RL (HL)	Rotate left loc.(HL) thru carry	4/15
RL (IX+d)	Rotate left loc.(IX+d) thru carry	6/23
RL (IY+d)	Rotate left loc.(IY+d) thru carry	6/23
RLCA	Rotate left circular Acc.	1/4
RLC r	Rotate left circular Reg.r	2/8
RLC (HL)	Rotate left circular loc.(HL)	4/15
RLC (IX+d)	Rotate left circular loc.(IX+d)	6/23
RLC (IY+d)	Rotate left circular loc.(IY+d)	6/23
RLD	Rotate digit left between Acc. and loc.(HL)	5/18
RRA	Rotate right Acc. thru carry	1/4
RR r	Rotate right Reg.r thru carry	2/8
RR (HL)	Rotate right loc.(HL) thru carry	4/15
RR (IX+d)	Rotate right loc.(IX+d) thru carry	6/23
RR (IY+d)	Rotate right loc.(IY+d) thru carry	6/23
RRCA	Rotate right circular Acc.	1/4
RRC r	Rotate right circular Reg.r	2/8
RRC (HL)	Rotate right circular loc.(HL)	4/15
RRC (IX+d)	Rotate right circular loc.(IX+d)	6/23
RRC (IY+d)	Rotate right circular loc.(IY+d)	6/23
RRD	Rotate digit right between Acc. and loc.(HL) ...	5/18
RST p	Restart to location p	3/11
SBC A,n	Subtract with carry value n from Acc.	2/7
SBC A,r	Subtract with carry Reg.r from Acc.	1/4
SBC A,(HL)	Subtract with carry loc.(HL) from Acc.	2/7
SBC A,(IX+d)	Subtract with carry loc.(IX+d) from Acc.	5/19
SBC A,(IY+d)	Subtract with carry loc.(IY+d) from Acc.	5/19
SBC HL,rr	Subtract with carry Reg.pair rr from HL	4/15
SCF	Set carry flag	1/4
SET b,r	Set Bit b of Reg.r	2/8
SET b,(HL)	Set Bit b of loc.(HL)	4/15
SET b,(IX+d)	Set Bit b of loc.(IX+d)	6/23
SET b,(IY+d)	Set Bit b of loc.(IY+d)	6/23

SLA r	Shift left arithmetic Reg.r	2/8
SLA (HL)	Shift left arithmetic loc.(HL)	4/15
SLA (IX+d)	Shift left arithmetic loc.(IX+d)	6/23
SLA (IY+d)	Shift left arithmetic loc.(IY+d)	6/23
SRA r	Shift right arithmetic Reg.r	2/8
SRA (HL)	Shift right arithmetic loc.(HL)	4/15
SRA (IX+d)	Shift right arithmetic loc.(IX+d)	6/23
SRA (IY+d)	Shift right arithmetic loc.(IY+d)	6/23
SRL r	Shift right logical Reg.r	2/8
SRL (HL)	Shift right logical loc.(HL)	4/15
SRL (IX+d)	Shift right logical loc.(IX+d)	6/23
SRL (IY+d)	Shift right logical loc.(IY+d)	6/23
SUB n	Subtract value n from Acc.	2/7
SUB r	Subtract Reg.r from Acc.	1/4
SUB (HL)	Subtract loc.(HL) from Acc.	2/7
SUB (IX+d)	Subtract loc.(IX+d) from Acc.	5/19
SUB (IY+d)	Subtract loc.(IY+d) from Acc.	5/19
XOR n	'EXCLUSIVE-OR' value n and Acc.	2/7
XOR r	'EXCLUSIVE-OR' Reg.r and Acc.	1/4
XOR (HL)	'EXCLUSIVE-OR' loc.(HL) and Acc.	2/7
XOR (IX+d)	'EXCLUSIVE-OR' loc.(IX+d) and Acc.	5/19
XOR (IY+d)	'EXCLUSIVE-OR' loc.(IY+d) and Acc.	5/19

13.4) Die Flag-Beeinflussung durch die Z80-Befehle

Nachstehend sind diejenigen Befehle des Z80 zusammengestellt, welche die Bedingungsbits (Flags) beeinflussen können. Es wird dabei für jeden dieser Befehle die Art der Beeinflussung der einzelnen Flags angegeben. (Bezüglich der Bedeutung der Flags wird auf Seite 165 verwiesen.)

Befehle	Bedingungsbits C	Z	P/V	S	N	H
ADD A,s ;ADC A,s	*	*	V	*	0	*
SUB s ;SBC A,s ;NEG	*	*	V	*	1	*
AND s	0	*	P	*	0	1
OR s ;XOR s	0	*	P	*	0	0
INC s	-	*	V	*	0	*
DEC m	-	*	V	*	1	*
ADD rr,ss	*	-	-	-	0	x
ADC HL,ss	*	*	V	*	0	x
SBC HL,ss	*	*	V	*	1	x
RLA ;RLCA ;RRA ;RRCA	*	-	-	-	0	0
RL m ;RLC m ;RR m ;RRC m SLA m ;SRA m ;SRL m	*	*	P	*	0	0
RLD ;RRD	-	*	P	*	0	0
DAA	*	*	P	*	-	*
CPL	-	-	-	-	1	1
SCF	1	-	-	-	0	0
CCF	*	-	-	-	0	x
IN r,(C)	-	*	P	*	0	0
INI ;IND ;OUTI ;OUTD	-	*	x	x	1	x
INIR ;INDR ;OTIR ;OTDR	-	1	x	x	1	x
LDI ;LDD	-	x	*	x	0	0
LDIR ;LDDR	-	x	0	x	0	0
CPI ;CPIR ;CPD ;CPDR	-	*	*	*	1	x
LD A,I ;LD A,R	-	*	IFF	*	0	0
BIT b,s	-	*	x	x	0	1
NEG	*	*	V	*	1	*

Erklärung:

* das Flag wird je nach Ergebnis der Operation beeinflußt
\- das Flag bleibt unbeeinflußt
0 das Flag wird 0 gesetzt
1 das Flag wird 1 gesetzt
x das Flag ist unbestimmt
V das P/V-Flag wird entsprechend dem Überlauf (Overflow) beeinflußt
P das P/V-Flag wird entsprechend der Parität beeinflußt
IFF das P/V-Flag übernimmt den Zustand des Interrupt-Flipflops

13.5) Hexadezimale Reihenfolge der Z80-Befehle

Auf den nächsten beiden Seiten sind sämtliche autorisierten Z80-Befehle nochmals zusammengestellt, wobei die Anordnung nach der numerischen, hexadezimalen Reihenfolge des Maschinencodes der Befehle erfolgte.

Die ersten 256 Befehle dieser Liste, mit den 1-Byte-Operationscodes 00 bis FF, stellen dabei den alten 8080-Befehlssatz dar, der ja im Z80-Befehlssatz mit unverändertem Maschinencode enthalten ist (mit Ausnahme der acht Befehle 08, 10, 18, 20, 28, 30, 38, D9, die im 8080-Befehlssatz nicht definiert sind und beim Z80-Befehlssatz eine eigene Bedeutung besitzen).

HEXADEZIMALE REIHENFOLGE DER Z80-BEFEHLE

00	NOP	3B	DEC SP	76	HALT	B1	OR C	EC....	CALL PE,NN	CB27	SLA A
01....	LD BC,NN	3C	INC A	77	LD (HL),A	B2	OR D	--		CB28	SRA B
02	LD (BC),A	3D	DEC A	78	LD A,B	B3	OR E	EE..	XOR N	CB29	SRA C
03	INC BC	3E..	LD A,N	79	LD A,C	B4	OR H	EF	RST 28	CB2A	SRA D
04	INC B	3F	CCF	7A	LD A,D	B5	OR L	F0	RET P	CB2B	SRA E
05	DEC B	40	LD B,B	7B	LD A,E	B6	OR (HL)	F1	POP AF	CB2C	SRA H
06..	LD B,N	41	LD B,C	7C	LD A,H	B7	OR A	F2....	JP P,NN	CB2D	SRA L
07	RLCA	42	LD B,D	7D	LD A,L	B8	CP B	F3	DI	CB2E	SRA (HL)
08	EX AF,AF'	43	LD B,E	7E	LD A,(HL)	B9	CP C	F4....	CALL P,NN	CB2F	SRA A
09	ADD HL,BC	44	LD B,H	7F	LD A,A	BA	CP D	F5	PUSH AF	CB38	SRL B
0A	LD A,(BC)	45	LD B,L	80	ADD A,B	BB	CP E	F6..	OR N	CB39	SRL C
0B	DEC BC	46	LD B,(HL)	81	ADD A,C	BC	CP H	F7	RST 30	CB3A	SRL D
0C	INC C	47	LD B,A	82	ADD A,D	BD	CP L	F8	RET M	CB3B	SRL E
0D	DEC C	48	LD C,B	83	ADD A,E	BE	CP (HL)	F9	LD SP,HL	CB3C	SRL H
0E..	LD C,N	49	LD C,C	84	ADD A,H	BF	CP A	FA....	JP M,NN	CB3D	SRL L
0F	RRCA	4A	LD C,D	85	ADD A,L	C0	RET NZ	FB	EI	CB3E	SRL (HL)
10..	DJNZ N	4B	LD C,E	86	ADD A,(HL)	C1	POP BC	FC....	CALL M,NN	CB3F	SRL A
11....	LD DE,NN	4C	LD C,H	87	ADD A,A	C2....	JP NZ,NN	--		CB40	BIT 0,B
12	LD (DE),A	4D	LD C,L	88	ADC A,B	C3....	JP NN	FE..	CP N	CB41	BIT 0,C
13	INC DE	4E	LD C,(HL)	89	ADC A,C	C4....	CALL NZ,NN	FF	RST 38	CB42	BIT 0,D
14	INC D	4F	LD C,A	8A	ADC A,D	C5	PUSH BC	CB00	RLC B	CB43	BIT 0,E
15	DEC D	50	LD D,B	8B	ADC A,E	C6..	ADD A,N	CB01	RLC C	CB44	BIT 0,H
16..	LD D,N	51	LD D,C	8C	ADC A,H	C7	RST 0	CB02	RLC D	CB45	BIT 0,L
17	RLA	52	LD D,D	8D	ADC A,L	C8	RET Z	CB03	RLC E	CB46	BIT 0,(HL)
18..	JR N	53	LD D,E	8E	ADC A,(HL)	C9	RET	CB04	RLC H	CB47	BIT 0,A
19	ADD HL,DE	54	LD D,H	8F	ADC A,A	CA....	JP Z,NN	CB05	RLC L	CB48	BIT 1,B
1A	LD A,(DE)	55	LD D,L	90	SUB B	--		CB06	RLC (HL)	CB49	BIT 1,C
1B	DEC DE	56	LD D,(HL)	91	SUB C	CC....	CALL Z,NN	CB07	RLC A	CB4A	BIT 1,D
1C	INC E	57	LD D,A	92	SUB D	CD....	CALL NN	CB08	RRC B	CB4B	BIT 1,E
1D	DEC E	58	LD E,B	93	SUB E	CE..	ADC A,N	CB09	RRC C	CB4C	BIT 1,H
1E..	LD E,N	59	LD E,C	94	SUB H	CF	RST 8	CB0A	RRC D	CB4D	BIT 1,L
1F	RRA	5A	LD E,D	95	SUB L	D0	RET NC	CB0B	RRC E	CB4E	BIT 1,(HL)
20..	JR NZ,N	5B	LD E,E	96	SUB (HL)	D1	POP DE	CB0C	RRC H	CB4F	BIT 1,A
21....	LD HL,NN	5C	LD E,H	97	SUB A	D2....	JP NC,NN	CB0D	RRC L	CB50	BIT 2,B
22....	LD (NN),HL	5D	LD E,L	98	SBC A,B	D3..	OUT N,A	CB0E	RRC (HL)	CB51	BIT 2,C
23	INC HL	5E	LD E,(HL)	99	SBC A,C	D4....	CALL NC,NN	CB0F	RRC A	CB52	BIT 2,D
24	INC H	5F	LD E,A	9A	SBC A,D	D5	PUSH DE	CB10	RL B	CB53	BIT 2,E
25	DEC H	60	LD H,B	9B	SBC A,E	D6..	SUB N	CB11	RL C	CB54	BIT 2,H
26..	LD H,N	61	LD H,C	9C	SBC A,H	D7	RST 10	CB12	RL D	CB55	BIT 2,L
27	DAA	62	LD H,D	9D	SBC A,L	D8	RET C	CB13	RL E	CB56	BIT 2,(HL)
28..	JR Z,N	63	LD H,E	9E	SBC A,(HL)	D9	EXX	CB14	RL H	CB57	BIT 2,A
29	ADD HL,HL	64	LD H,H	9F	SBC A,A	DA....	JP C,NN	CB15	RL L	CB58	BIT 3,B
2A....	LD HL,(NN)	65	LD H,L	A0	AND B	DB..	IN A,N	CB16	RL (HL)	CB59	BIT 3,C
2B	DEC HL	66	LD H,(HL)	A1	AND C	DC....	CALL C,NN	CB17	RL A	CB5A	BIT 3,D
2C	INC L	67	LD H,A	A2	AND D	--		CB18	RR B	CB5B	BIT 3,E
2D	DEC L	68	LD L,B	A3	AND E	DE..	SBC A,N	CB19	RR C	CB5C	BIT 3,H
2E..	LD L,N	69	LD L,C	A4	AND H	DF	RST 18	CB1A	RR D	CB5D	BIT 3,L
2F	CPL	6A	LD L,D	A5	AND L	E0	RET PO	CB1B	RR E	CB5E	BIT 3,(HL)
30..	JR NC,N	6B	LD L,E	A6	AND (HL)	E1	POP HL	CB1C	RR H	CB5F	BIT 3,A
31....	LD SP,NN	6C	LD L,H	A7	AND A	E2....	JP PO,NN	CB1D	RR L	CB60	BIT 4,B
32....	LD (NN),A	6D	LD L,L	A8	XOR B	E3	EX (SP),HL	CB1E	RR (HL)	CB61	BIT 4,C
33	INC SP	6E	LD L,(HL)	A9	XOR C	E4....	CALL PO,NN	CB1F	RR A	CB62	BIT 4,D
34	INC (HL)	6F	LD L,A	AA	XOR D	E5	PUSH HL	CB20	SLA B	CB63	BIT 4,E
35	DEC (HL)	70	LD (HL),B	AB	XOR E	E6..	AND N	CB21	SLA C	CB64	BIT 4,H
36..	LD (HL),N	71	LD (HL),C	AC	XOR H	E7	RST 20	CB22	SLA D	CB65	BIT 4,L
37	SCF	72	LD (HL),D	AD	XOR L	E8	RET PE	CB23	SLA E	CB66	BIT 4,(HL)
38..	JR C,N	73	LD (HL),E	AE	XOR (HL)	E9	JP (HL)	CB24	SLA H	CB67	BIT 4,A
39	ADD HL,SP	74	LD (HL),H	AF	XOR A	EA....	JP PE,NN	CB25	SLA L	CB68	BIT 5,B
3A....	LD A,(NN)	75	LD (HL),L	B0	OR B	EB	EX DE,HL	CB26	SLA (HL)	CB69	BIT 5,C

```
CB6A    BIT 5,D
CB6B    BIT 5,E
CB6C    BIT 5,H
CB6D    BIT 5,L
CB6E    BIT 5,(HL)
CB6F    BIT 5,A
CB70    BIT 6,B
CB71    BIT 6,C
CB72    BIT 6,D
CB73    BIT 6,E
CB74    BIT 6,H
CB75    BIT 6,L
CB76    BIT 6,(HL)
CB77    BIT 6,A
CB78    BIT 7,B
CB79    BIT 7,C
CB7A    BIT 7,D
CB7B    BIT 7,E
CB7C    BIT 7,H
CB7D    BIT 7,L
CB7E    BIT 7,(HL)
CB7F    BIT 7,A
CB80    RES 0,B
CB81    RES 0,C
CB82    RES 0,D
CB83    RES 0,E
CB84    RES 0,H
CB85    RES 0,L
CB86    RES 0,(HL)
CB87    RES 0,A
CB88    RES 1,B
CB89    RES 1,C
CB8A    RES 1,D
CB8B    RES 1,E
CB8C    RES 1,H
CB8D    RES 1,L
CB8E    RES 1,(HL)
CB8F    RES 1,A
CB90    RES 2,B
CB91    RES 2,C
CB92    RES 2,D
CB93    RES 2,E
CB94    RES 2,H
CB95    RES 2,L
CB96    RES 2,(HL)
CB97    RES 2,A
CB98    RES 3,B
CB99    RES 3,C
CB9A    RES 3,D
CB9B    RES 3,E
CB9C    RES 3,H
CB9D    RES 3,L
CB9E    RES 3,(HL)
CB9F    RES 3,A
CBA0    RES 4,B
CBA1    RES 4,C
CBA2    RES 4,D
CBA3    RES 4,E
CBA4    RES 4,H

CBA5    RES 4,L
CBA6    RES 4,(HL)
CBA7    RES 4,A
CBA8    RES 5,B
CBA9    RES 5,C
CBAA    RES 5,D
CBAB    RES 5,E
CBAC    RES 5,H
CBAD    RES 5,L
CBAE    RES 5,(HL)
CBAF    RES 5,A
CBB0    RES 6,B
CBB1    RES 6,C
CBB2    RES 6,D
CBB3    RES 6,E
CBB4    RES 6,H
CBB5    RES 6,L
CBB6    RES 6,(HL)
CBB7    RES 6,A
CBB8    RES 7,B
CBB9    RES 7,C
CBBA    RES 7,D
CBBB    RES 7,E
CBBC    RES 7,H
CBBD    RES 7,L
CBBE    RES 7,(HL)
CBBF    RES 7,A
CBC0    SET 0,B
CBC1    SET 0,C
CBC2    SET 0,D
CBC3    SET 0,E
CBC4    SET 0,H
CBC5    SET 0,L
CBC6    SET 0,(HL)
CBC7    SET 0,A
CBC8    SET 1,B
CBC9    SET 1,C
CBCA    SET 1,D
CBCB    SET 1,E
CBCC    SET 1,H
CBCD    SET 1,L
CBCE    SET 1,(HL)
CBCF    SET 1,A
CBD0    SET 2,B
CBD1    SET 2,C
CBD2    SET 2,D
CBD3    SET 2,E
CBD4    SET 2,H
CBD5    SET 2,L
CBD6    SET 2,(HL)
CBD7    SET 2,A
CBD8    SET 3,B
CBD9    SET 3,C
CBDA    SET 3,D
CBDB    SET 3,E
CBDC    SET 3,H
CBDD    SET 3,L
CBDE    SET 3,(HL)
CBDF    SET 3,A

CBE0    SET 4,B
CBE1    SET 4,C
CBE2    SET 4,D
CBE3    SET 4,E
CBE4    SET 4,H
CBE5    SET 4,L
CBE6    SET 4,(HL)
CBE7    SET 4,A
CBE8    SET 5,B
CBE9    SET 5,C
CBEA    SET 5,D
CBEB    SET 5,E
CBEC    SET 5,H
CBED    SET 5,L
CBEE    SET 5,(HL)
CBEF    SET 5,A
CBF0    SET 6,B
CBF1    SET 6,C
CBF2    SET 6,D
CBF3    SET 6,E
CBF4    SET 6,H
CBF5    SET 6,L
CBF6    SET 6,(HL)
CBF7    SET 6,A
CBF8    SET 7,B
CBF9    SET 7,C
CBFA    SET 7,D
CBFB    SET 7,E
CBFC    SET 7,H
CBFD    SET 7,L
CBFE    SET 7,(HL)
CBFF    SET 7,A
DD09      ADD IX,BC
DD19      ADD IX,DE
DD21....  LD IX,NN
DD22....  LD (NN),IX
DD23      INC IX
DD29      ADD IX,IX
DD2A....  LD IX,(NN)
DD2B      DEC IX
DD34..    INC (IX+N)
DD35..    DEC (IX+N)
DD36....  LD (IX+N),N
DD39      ADD IX,SP
DD46..    LD B,(IX+N)
DD4E..    LD C,(IX+N)
DD56..    LD D,(IX+N)
DD5E..    LD E,(IX+N)
DD66..    LD H,(IX+N)
DD6E..    LD L,(IX+N)
DD70..    LD (IX+N),B
DD71..    LD (IX+N),C
DD72..    LD (IX+N),D
DD73..    LD (IX+N),E
DD74..    LD (IX+N),H
DD75..    LD (IX+N),L
DD77..    LD (IX+N),A
DD7E..    LD A,(IX+N)
DD86..    ADD A,(IX+N)

DD8E..    ADC A,(IX+N)
DD96..    SUB (IX+N)
DD9E..    SBC A,(IX+N)
DDA6..    AND (IX+N)
DDAE..    XOR (IX+N)
DDB6..    OR (IX+N)
DDBE..    CP (IX+N)
DDE1      POP IX
DDE3      EX (SP),IX
DDE5      PUSH IX
DDE9      JP (IX)
DDF9      LD SP,IX
DDCB..06  RLC (IX+N)
DDCB..0E  RRC (IX+N)
DDCB..16  RL (IX+N)
DDCB..1E  RR (IX+N)
DDCB..26  SLA (IX+N)
DDCB..2E  SRA (IX+N)
DDCB..3E  SRL (IX+N)
DDCB..46  BIT 0,(IX+N)
DDCB..4E  BIT 1,(IX+N)
DDCB..56  BIT 2,(IX+N)
DDCB..5E  BIT 3,(IX+N)
DDCB..66  BIT 4,(IX+N)
DDCB..6E  BIT 5,(IX+N)
DDCB..76  BIT 6,(IX+N)
DDCB..7E  BIT 7,(IX+N)
DDCB..86  RES 0,(IX+N)
DDCB..8E  RES 1,(IX+N)
DDCB..96  RES 2,(IX+N)
DDCB..9E  RES 3,(IX+N)
DDCB..A6  RES 4,(IX+N)
DDCB..AE  RES 5,(IX+N)
DDCB..B6  RES 6,(IX+N)
DDCB..BE  RES 7,(IX+N)
DDCB..C6  SET 0,(IX+N)
DDCB..CE  SET 1,(IX+N)
DDCB..D6  SET 2,(IX+N)
DDCB..DE  SET 3,(IX+N)
DDCB..E6  SET 4,(IX+N)
DDCB..EE  SET 5,(IX+N)
DDCB..F6  SET 6,(IX+N)
DDCB..FE  SET 7,(IX+N)
ED40      IN B,(C)
ED41      OUT (C),B
ED42      SBC HL,BC
ED43....  LD (NN),BC
ED44      NEG
ED45      RETN
ED46      IM 0
ED47      LD I,A
ED48      IN C,(C)
ED49      OUT (C),C
ED4A      ADC HL,BC
ED4B....  LD BC,(NN)
ED4D      RETI
ED50      IN D,(C)
ED51      OUT (C),D
ED52      SBC HL,DE

ED53....  LD (NN),DE
ED56      IM 1
ED57      LD A,I
ED58      IN E,(C)
ED59      OUT (C),E
ED5A      ADC HL,DE
ED5B....  LD DE,(NN)
ED5E      IM 2
ED60      IN H,(C)
ED61      OUT (C),H
ED62      SBC HL,HL
ED67      RRD
ED68      IN L,(C)
ED69      OUT (C),L
ED6A      ADC HL,HL
ED6F      RLD
ED72      SBC HL,SP
ED73....  LD (NN),SP
ED78      IN A,(C)
ED79      OUT (C),A
ED7A      ADC HL,SP
ED7B....  LD SP,(NN)
EDA0      LDI
EDA1      CPI
EDA2      INI
EDA3      OUTI
EDA8      LDD
EDA9      CPD
EDAA      IND
EDAB      OUTD
EDB0      LDIR
EDB1      CPIR
EDB2      INIR
EDB3      OTIR
EDB8      LDDR
EDB9      CPDR
EDBA      INDR
EDBB      OTDR
FD09      ADD IY,BC
FD19      ADD IY,DE
FD21....  LD IY,NN
FD22....  LD (NN),IY
FD23      INC IY
FD29      ADD IY,IY
FD2A....  LD IY,(NN)
FD2B      DEC IY
FD34..    INC (IY+N)
FD35..    DEC (IY+N)
FD36....  LD (IY+N),N
FD39      ADD IY,SP
FD46..    LD B,(IY+N)
FD4E..    LD C,(IY+N)
FD56..    LD D,(IY+N)
FD5E..    LD E,(IY+N)
FD66..    LD H,(IY+N)
FD6E..    LD L,(IY+N)
FD70..    LD (IY+N),B
FD71..    LD (IY+N),C
FD72..    LD (IY+N),D

FD73..    LD (IY+N),E
FD74..    LD (IY+N),H
FD75..    LD (IY+N),L
FD77..    LD (IY+N),A
FD7E..    LD A,(IY+N)
FD86..    ADD A,(IY+N)
FD8E..    ADC A,(IY+N)
FD96..    SUB (IY+N)
FD9E..    SBC A,(IY+N)
FDA6..    AND (IY+N)
FDAE..    XOR (IY+N)
FDB6..    OR (IY+N)
FDBE..    CP (IY+N)
FDE1      POP IY
FDE3      EX (SP),IY
FDE5      PUSH IY
FDE9      JP (IY)
FDF9      LD SP,IY
FDCB..06  RLC (IY+N)
FDCB..0E  RRC (IY+N)
FDCB..16  RL (IY+N)
FDCB..1E  RR (IY+N)
FDCB..26  SLA (IY+N)
FDCB..2E  SRA (IY+N)
FDCB..3E  SRL (IY+N)
FDCB..46  BIT 0,(IY+N)
FDCB..4E  BIT 1,(IY+N)
FDCB..56  BIT 2,(IY+N)
FDCB..5E  BIT 3,(IY+N)
FDCB..66  BIT 4,(IY+N)
FDCB..6E  BIT 5,(IY+N)
FDCB..76  BIT 6,(IY+N)
FDCB..7E  BIT 7,(IY+N)
FDCB..86  RES 0,(IY+N)
FDCB..8E  RES 1,(IY+N)
FDCB..96  RES 2,(IY+N)
FDCB..9E  RES 3,(IY+N)
FDCB..A6  RES 4,(IY+N)
FDCB..AE  RES 5,(IY+N)
FDCB..B6  RES 6,(IY+N)
FDCB..BE  RES 7,(IY+N)
FDCB..C6  SET 0,(IY+N)
FDCB..CE  SET 1,(IY+N)
FDCB..D6  SET 2,(IY+N)
FDCB..DE  SET 3,(IY+N)
FDCB..E6  SET 4,(IY+N)
FDCB..EE  SET 5,(IY+N)
FDCB..F6  SET 6,(IY+N)
FDCB..FE  SET 7,(IY+N)
```

13.6) Anhang: Äquivalente Befehle der Mikroprozessoren 8080 und Z80

Wie bereits eingangs erwähnt wurde, ist der Mikroprozessor Z80 "aufwärtskompatibel" zum Mikroprozessor 8080, d.h. der Befehlssatz des Mikroprozessors 8080 ist im Befehlssatz des Mikroprozessors Z80 enthalten. Allerdings gilt diese Kompatibilität nur auf der (entscheidenden) Ebene des Maschinencodes, während auf ASSEMBLER-Ebene die mnemonischen Abkürzungen der Befehle zwischen den beiden Mikroprozessoren Unterschiede aufweisen. In der folgenden Tabelle sind die äquivalenten Befehle der Mikroprozessoren 8080 und Z80 in ihren verschiedenen ASSEMBLER-Schreibweisen einander gegenübergestellt, der Maschinencode, d.h. die Hexadezimal- bzw. Binär-Darstellung dieser Befehle ist identisch.

Es sind also Maschinen-Programme, welche nur aus diesen Befehlen aufgebaut wurden, in allen drei Mikrocomputersystemen 8080, 8085, und Z80 gleichermaßen lauffähig!

Äquivalente Befehle: 8080 - Z80

8080	Z80
ACI d8	ADC A,n
ADC M	ADC A,(HL)
ADC r	ADC A,r
ADD M	ADD A,(HL)
ADD r	ADD A,r
ADI d8	ADD A,n
ANA M	AND (HL)
ANA r	AND r
ANI d8	AND n
CALL adr	CALL nn
CC adr	CALL C,nn
CM adr	CALL M,nn
CMA	CPL
CMC	CCF
CMP M	CP (HL)
CMP r	CP r
CNC adr	CALL NC,nn
CNZ adr	CALL NZ,nn
CP adr	CALL P,nn
CPE adr	CALL PE,nn
CPI d8	CP n
CPO adr	CALL PO,nn
CZ adr	CALL Z,nn
DAA	DAA
DAD B	ADD HL,BC
DAD D	ADD HL,DE
DAD H	ADD HL,HL
DAD SP	ADD HL,SP
DCR M	DEC (HL)
DCR r	DEC r
DCX B	DEC BC
DCX D	DEC DE
DCX H	DEC HL
DCX SP	DEC SP
DI	DI
EI	EI
HLT	HALT
IN n	IN A,(n)
INR M	INC (HL)
INR r	INC r
INX B	INC BC
INX D	INC DE
INX H	INC HL
INX SP	INC SP
JC adr	JP C,nn
JM adr	JP M,nn
JMP adr	JP nn
JNC adr	JP NC,nn
JNZ adr	JP NZ,nn
JP adr	JP P,nn
JPE adr	JP PE,nn
JPO adr	JP PO,nn
JZ adr	JP Z,nn
LDA adr	LD A,(nn)
LDAX B	LD A,(BC)
LDAX D	LD A,(DE)
LHLD adr	LD HL,(nn)
LXI B,d16	LD BC,nn
LXI D,d16	LD DE,nn
LXI H,d16	LD HL,nn
LXI SP,d16	LD SP,nn
MOV M,r	LD (HL),r
MOV r,M	LD r,(HL)
MOV r1,r2	LD r1,r2
MVI M,d8	LD (HL),n
MVI r,d8	LD r,n
NOP	NOP
ORA M	OR (HL)
ORA r	OR r
ORI d8	OR n
OUT n	OUT (n),A
PCHL	JP (HL)
POP B	POP BC
POP D	POP DE
POP H	POP HL
POP PSW	POP AF
PUSH B	PUSH BC
PUSH D	PUSH DE
PUSH H	PUSH HL
PUSH PSW	PUSH AF
RAL	RLA
RAR	RRA
RC	RET C
RET	RET
RLC	RLCA
RM	RET M
RNC	RET NC
RNZ	RET NZ
RP	RET P
RPE	RET PE
RPO	RET PO
RRC	RRCA
RST d	RST n
RZ	RET Z
SBB M	SBC A,(HL)
SBB r	SBC A,r
SBI d8	SBC A,n
SHLD adr	LD (nn),HL
SPHL	LD SP,HL
STA adr	LD (nn),A
STAX B	LD (BC),A
STAX D	LD (DE),A
STC	SCF
SUB M	SUB (HL)
SUB r	SUB r
SUI d8	SUB n
XCHG	EX DE,HL
XRA M	XOR (HL)
XRA r	XOR r
XRI d8	XOR n
XTHL	EX (SP),HL

14) AUFBAU VON MIKROCOMPUTERN MIT DEM MIKROPROZESSOR Z80

Beim Aufbau von Mikrocomputern mit dem Mikroprozessor Z80 als CPU kann man weitgehend auf die Standard-Bausteine zurückgreifen, die wir beim System 8080 kennengelernt hatten. Insbesondere werden die Standard-Speicher-Bausteine (RAM, ROM, EPROM), die wir dort verwendet haben, unmittelbar auch im System Z80 eingesetzt. Auch die bekannten Standard-Ein/Ausgabe-Bausteine (wie Parallel-Ein/Ausgabe (PIO) 8255 und Serien-Ein/Ausgabe (USART) 8251) werden häufig beim Aufbau von Z80-Mikrocomputersystemen übernommen. Daneben wurden für den Mikroprozessor Z80 spezielle Ein/Ausgabe-Bausteine entwickelt, die auf die Besonderheiten dieses Prozessors und seine spezifischen Interruptmöglichkeiten abgestimmt sind. Wir werden im folgenden diese verschiedenen Möglichkeiten beim Aufbau von Mikrocomputern mit Z80-CPU behandeln.

14.1) Eine Z80-CPU mit den 8080-Steuerleitungen $\overline{MEMR}$,$\overline{MEMW}$,$\overline{IOR}$,$\overline{IOW}$

Wie wir sahen, haben die 4 Hauptsteuerleitungen beim Mikroprozessor Z80 etwas andere Funktionen als beim Mikroprozessor 8080. Wir werden zunächst versuchen, die Z80-Steuersignale in die alten 8080-Signale überzuführen. Obwohl die dadurch gewonnene Schaltung (Abb. 29) nur geringe praktische Bedeutung besitzt, werden wir hier mit ihr beginnen, weil sie unmittelbar den didaktischen Anschluß an die früheren Mikrocomputer-Schaltungen erlaubt, die wir beim 8080-System kennengelernt hatten.

Die 4 Low-aktiven Z80-Steuersignale $\overline{RD}$, $\overline{WR}$, $\overline{MREQ}$, $\overline{IORQ}$ sind durch 4 einfache ODER-Verknüpfungen in die 4 Low-aktiven 8080-Steuersignale $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$, $\overline{IOW}$ umwandelbar, wie Abb. 29 zeigt: z.B. bedeutet beim Z80 gleichzeitiges aktives Lesesignal $\overline{RD}$(=0) und aktives Speicherauswahlsignal $\overline{MREQ}$(=0), daß wir es mit einem Speicherlesesignal, d.h. mit $\overline{MEMR}$=0 zu tun haben; die eingezeichnete

ODER-Verknüpfung beider Leitungen bewirkt, daß die neu gebildete Steuerleitung $\overline{MEMR}$ in diesem Fall (und nur in diesem Fall) gleich Null wird.

Mit der Schaltung von Abb. 29 haben wir somit eine Z80-CPU gewonnen, die unmittelbar und ohne Einschränkung die frühere 8080-CPU (bestehend aus den 3 Bausteinen 8080, 8224 und 8228) ersetzen kann, so daß wir die dort beschriebenen Mikrocomputer-Schaltungen ohne Veränderungen übernehmen und sie mit der höher integrierten und leistungsfähigeren Z80-CPU ausstatten können.

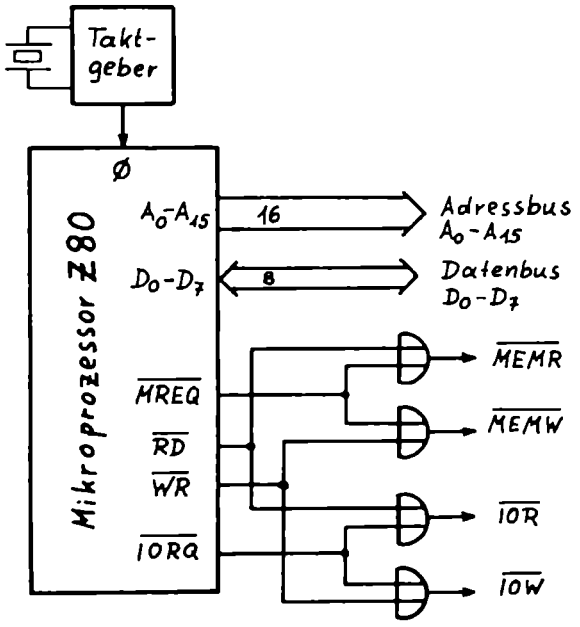


Abb.29: Z80-CPU mit den 8080-Steuerleitungen $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$, $\overline{IOW}$

14.2) Mikrocomputer mit Z80-CPU und Standard-Ein/Ausgabe-Bausteinen

Es ist natürlich im allgemeinen nicht notwendig, daß man auf die soeben beschriebene modifizierte Z80-CPU übergeht, wenn man Mikrocomputer mit dem Mikroprozessor Z80 aufbauen will. Vielmehr wird man unmittelbar die Steuerleitungen $\overline{RD}$, $\overline{WR}$, $\overline{MREQ}$ und $\overline{IORQ}$ des Mikroprozessors Z80 verwenden, um die Speicher- und Ein/Ausgabe-Bausteine des Mikrocomputers anzusteuern. Als Speicherbausteine werden dabei durchweg die bekannten Standard-RAM, -ROM und -EPROM-Bausteine verwendet, bei der Ein/Ausgabe-Einheit hat man die Wahl zwischen Standard-E/A-Bausteinen und speziellen Z80-E/A-Bausteinen. Wir werden zunächst in diesem Kapitel versuchen, Mikrocomputerschaltungen mit den uns schon vertrauten Standard-Ein/Ausgabe-Bausteinen (z.B. PIO 8255 und USART 8251) aufzubauen, was uns zu der Schaltung Abb. 30 führt.

Zum Verständnis des Verdrahtungsschemas der Schaltung Abb. 30 muß man sich nochmals die Bedeutung der 4 wesentlichen Z80-Steuersignale klarmachen: Es bedeutet $\overline{RD}$ das allgemeine Lesesignal (für Speicher und E/A), $\overline{WR}$ das allgemeine Schreibsignal (für Speicher und E/A), $\overline{MREQ}$ das aktivierende Signal für die Speicherbausteine, und $\overline{IORQ}$ das aktivierende Signal für die E/A-Bausteine. Die Unterscheidung zwischen Speicher- und Ein/Ausgabe-Einheit erfolgt also über 2 getrennte Leitungen $\overline{MREQ}$ und $\overline{IORQ}$, demzufolge ist in Abb. 30 die (Low-aktive) $\overline{MREQ}$-Leitung mit dem Bausteinfreigabeeingang $\overline{CE}$ der Speicherbausteine verbunden, entsprechend die $\overline{IORQ}$-Leitung mit dem $\overline{CE}$-Eingang der E/A-Bausteine.
Für die Anbringung der Baustein-Auswahl-Leitungen (Chip-Select) wird bei den Speicher- und E/A-Bausteinen ein weiterer Bausteinfreigabeeingang (CE bzw. CS) erforderlich (bzw. eine vorgeschaltete Logik bei nur einem einzigen CE-Anschluß). Bei größeren, aus mehreren Bausteinen aufgebauten Speichern erfolgt die Bausteinauswahl im allgemeinen wie in Abb. 30 über eine Chip-Select-Logik, bei der in bekannter Weise einige höhere Adressbits zu

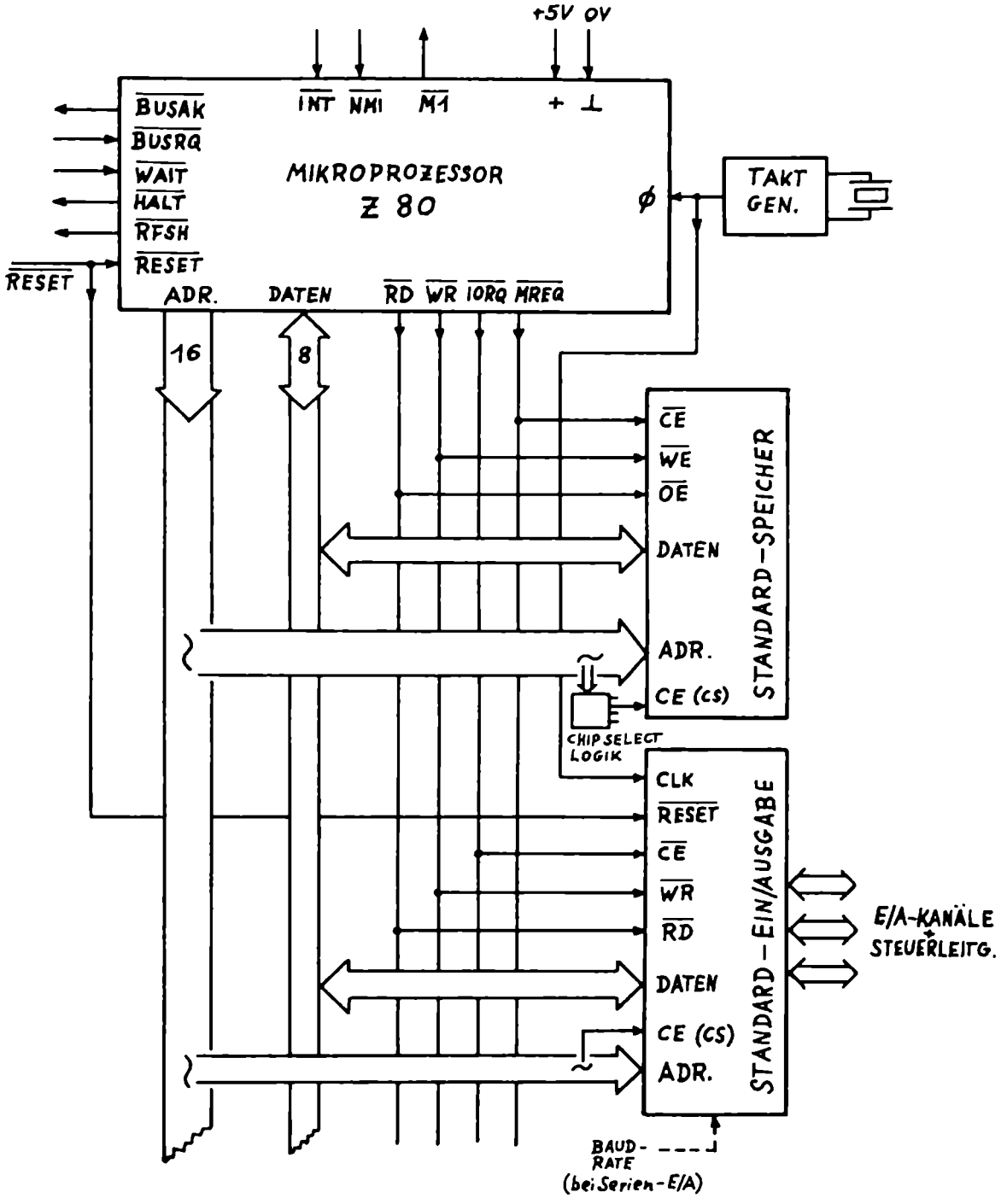


Abb.30: Mikrocomputer Z80 mit Standard Speicher- und E/A-Bausteinen

Chip-Select-Signalen dekodiert werden. Bei der Auswahl der Ein/Ausgabe-Bausteine wird man wieder am einfachsten mit Hilfe der "linearen Auswahl" (Kapitel 5.32) zum Ziel kommen.

Nach diesen Erläuterungen und mit den früheren Erkenntnissen aus den entsprechenden 8080-Schaltungen wird die in Abb. 30 dargestellte Mikrocomputer-Schaltung unmittelbar verständlich. Als "Standard-Speicher-Bausteine" kann man sich dabei die früheren Bausteine 2716 (EPROM) oder 6116 (RAM) von Kapitel 5 eingesetzt denken , als "Standard-E/A-Bausteine" die früheren Bausteine 8255-PIO (Parallel-Ein/Ausgabe) oder 8251-USART (Serien-Ein/Ausgabe). Im letzteren Falle müssen die Taktsignale für die serielle Datenübertragung (Baud-Rate-Signale) noch gesondert zugeführt werden. *)

Anmerkung: Für die Adressierung der E/A-Bausteine ist wieder sowohl die "isolierte Ein/Ausgabe" (Standard-E/A) als auch die "Speicher-Ein/Ausgabe" (Memory-Mapped-E/A) möglich (vgl.§ 5.32). Memory-Mapped-E/A ist beim System Z80 sogar einfacher als beim System 8080, weil $\overline{RD}$ und $\overline{WR}$ alle Bausteine gleichermaßen bedient, also die auswählende A15-Adressleitung (0 für Speicher, 1 für E/A) unmittelbar an die Chip-Select-Eingänge der Speicher- und E/A-Bausteine gelegt werden kann.

---

*) Es ist zu beachten, daß bei den schnellen Mikroprozessor-Versionen (z.B. Z80 H) auch die schnellen Versionen von Speicher- und E/A-Bausteinen eingesetzt werden müssen.

14.3) Spezielle Ein/Ausgabe-Bausteine für das Mikrocomputersystem Z80

Im Rahmen der Z80-Bausteinfamilie wurden spezielle Bausteine für den Datenaustausch mit der Peripherie entwickelt. Es sollen im folgenden 2 typische Bausteine kurz vorgestellt werden: die Parallelschnittstelle Z80-PIO und die serielle Schnittstelle Z80-SIO. Wegen der Besonderheiten des Z80-Prozessors und seiner spezifischen Steuersignale sind diese Bausteine nur mit diesem Mikroprozessor zu betreiben, wenn man ihre Möglichkeiten voll ausschöpfen will.
Insbesondere ist die in diesen Bausteinen integrierte Interrupt-Logik (mit der Möglichkeit, aus mehreren Bausteinen eine Interrupt-"Prioritätskette" zu bilden) auf den Interrupt-Modus 2 des Mikroprozessors Z80 speziell abgestimmt (siehe hierzu später).

Die erwähnten E/A-Bausteine können bezüglich ihrer Betriebsarten und verschiedenen Steuerungsvarianten auf komplizierte Weise programmiert werden (was oft nicht einfacher ist als den Mikroprozessor selbst zu programmieren). Eine vollständige Beschreibung dieser Bausteine und ihrer vielfältigen Programmierungsmöglichkeiten würde den Rahmen dieser Einführung übersteigen; es wird daher hier (wie schon früher bei den Standard PIO- und USART-Bausteinen) nur auf die grundsätzliche Funktionsweise eingegangen und im übrigen auf die ausführliche Literatur der Herstellerfirmen verwiesen (siehe Literaturverzeichnis am Schluß).

14.31) Der Parallel-Ein/Ausgabe-Baustein Z80-PIO

Die Z80-PIO (Parallel Input/Output) ist ein programmierbarer E/A-Baustein zur parallelen Datenübertragung zwischen Mikrocomputer und Peripherie. Der interne Aufbau der Z80-PIO ist vereinfacht im Blockschaltbild Abb. 31 wiedergegeben (vgl. hierzu auch den Aufbau unserer "Standard"-PIO 8255 auf S. 63). Die Z80-PIO besitzt

2 Kanäle A und B mit je 8 Leitungen, die über ein Betriebsartenregister in 4 Betriebsarten programmiert werden können:
8-Bit-parallele Eingabe, 8-Bit-parallele Ausgabe, bidirektionaler 8-Bit-Kanal, sowie Bitmodus, wobei im letzteren Fall jede einzelne Leitung individuell als Eingabe- oder Ausgabeleitung programmiert werden kann. (Näheres siehe Literatur)
Die ein- und ausgehenden Parallel-Daten werden in der PIO in Registern zwischengespeichert und der Datenaustausch mit den peripheren Geräten erfolgt im Quittungsbetrieb ("handshaking") über je 2 Handshake-Leitungen pro Kanal (READY und $\overline{\text{STROBE}}$). Somit ist ein Datenaustausch mit einem breiten Spektrum von peripheren Schaltungen ohne zusätzliche externe Logik möglich.

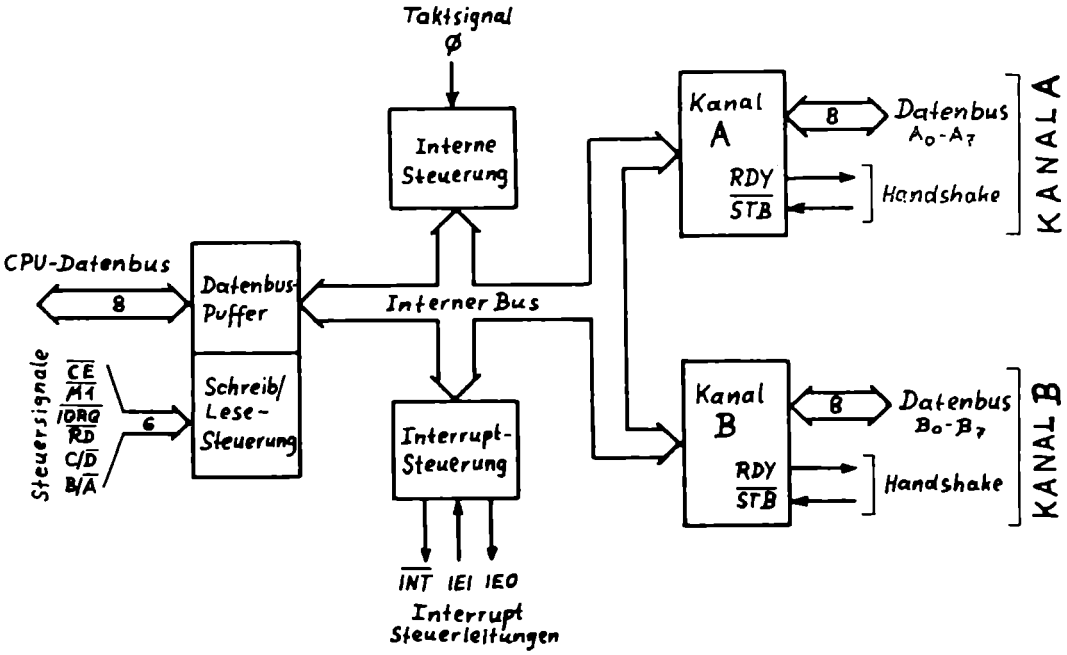


Abb.31: E/A-Baustein Z80-PIO für parallelen Datenaustausch

Die Z80-PIO ist (wie auch alle übrigen Peripheriebausteine der Z80-Familie) für die Bedienung von Interrupt-Anforderungen besonders eingerichtet. Hierzu besitzt jeder Kanal (A und B) der PIO ein "Interrupt-Vektor-Register" zur Aufnahme einer 8-Bit-Vektoradresse für die Abwicklung des Z80-Interrupt-Modus 2.

Wie wir bei der Besprechung der Z80-Interruptmöglichkeiten (S.173) gesehen hatten, erfordert die Abwicklung eines Interrupts nach dem Modus 2, daß das periphere Gerät, das die Unterbrechung anfordert, das niederwertige Byte einer Zeigeradresse an die CPU liefert. Dieses Byte wird vom Interrupt-Vektor-Register in der PIO bereitgestellt, in das es zuvor einprogrammiert werden muß. Jeder Kanal (A und B) besitzt ein unabhängiges Interrupt-Vektor- Register, wobei Unterbrechungen, welche über Kanal A angefordert werden, eine (schaltungsmäßig festgelegte) höhere Priorität besitzen als über Kanal B angeforderte Interrupts. Die Interrupt-Anforderung durch ein externes Gerät kann dabei, je nach programmierter Betriebsart, über einzelne Kanal-Leitungen oder über die Strobe-Leitung erfolgen.

Die Interrupt-Anforderung wird von der PIO über die $\overline{INT}$-Leitung an die CPU weitergeleitet; die Interrupt-Priorisierungsleitungen IEI (Interrupt Enable IN) und IEO (Interrupt Enable Out) werden wir später bei der Zusammenschaltung mehrerer E/A-Bausteine zu "Interrupt-Prioritätsketten" noch kennenlernen.

Anschluß der Z80-PIO an den Mikroprozessor Z80

Die nachfolgende Abb. 32 zeigt eine typische Schaltung zum Anschluß der Z80-PIO an die Z80-CPU:

- Die gleichlautenden Anschlüsse von CPU und PIO werden miteinander verbunden.
- Über den PIO-Eingang B/$\overline{A}$ erfolgt die Unterscheidung zwischen den beiden Kanälen A und B; im vorliegenden Fall wird diese Unterscheidung durch die Adresse A0 vorgenommen: B/$\overline{A}$=0 bedeutet Kanal A, B/$\overline{A}$=1 bedeutet Kanal B.

- Über den PIO-Eingang C/$\overline{D}$ wird festgelegt, ob Steuersignale (C/$\overline{D}$=1) oder Daten (C/$\overline{D}$=0) an die PIO übermittelt werden sollen; im vorliegenden Fall wird dies durch die Adresse A1 bewirkt.
- Die Adresse A7 liegt am PIO-Freigabeeingang $\overline{CE}$ und bewirkt die lineare Auswahl des Bausteins.

Mit dieser Schaltung stehen dem Anwender zwei 8-Bit-Kanäle A und B mit den zugehörigen Handshake-Leitungen RDY (Ready) und $\overline{STB}$ (Strobe = Übernahme) zu Verfügung, mit denen der parallele Datenaustausch mit peripheren Geräten vorgenommen werden kann.

Auffallend ist bei dieser Schaltung, daß die Z80-PIO keinen Anschluß für die $\overline{WR}$-Leitung des Mikroprozessors besitzt. Das Schreibsignal für die Z80-PIO wird durch das Fehlen des aktiven $\overline{RD}$-Signales in Zusammenwirken mit den Signalen C/$\overline{D}$ und $\overline{IORQ}$ gebildet!

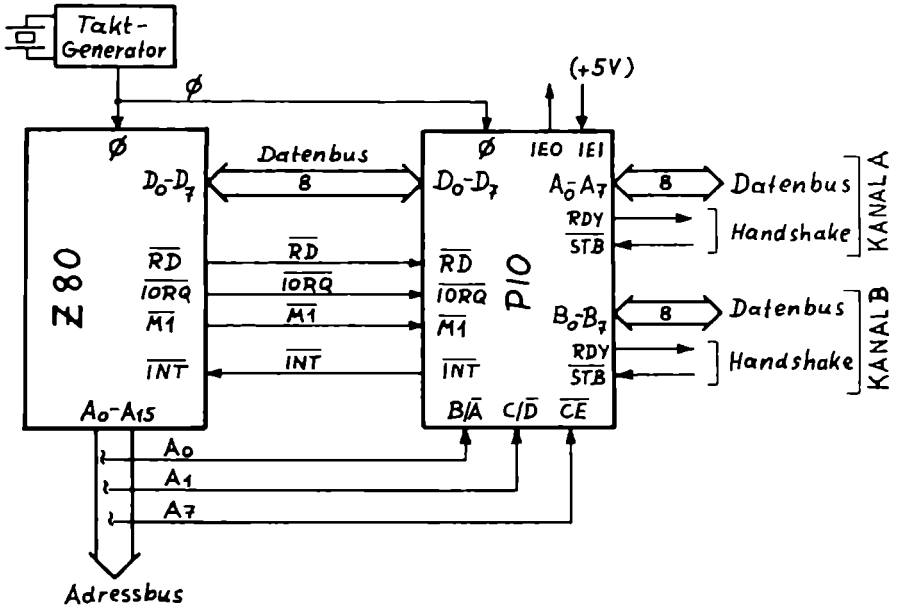


Abb. 32: Anschluss der Z80-PIO an den Mikroprozessor Z80

14.32) Der Serien-Ein/Ausgabe-Baustein Z80-SIO

Die Z80-SIO (Serial Input Output) ist ein programmierbarer E/A-Baustein zur seriellen Datenübertragung zwischen Mikrocomputer und Peripherie. Sie wandelt 8-Bit-parallele Daten, die aus dem Mikroprozessor kommen, über ein Zwischenregister in einen seriellen Datenstrom nach außen um, und umgekehrt werden von außen angelieferte serielle Daten in 8-Bit-parallele Daten für den Mikrocomputer umgeformt. Es handelt sich bei diesem Baustein um eine universelle Schnittstelle mit 2 unabhängigen Vollduplex-Kanälen, die sowohl für asynchronen als auch für synchronen seriellen Datenaustausch geeignet ist: USART (= Universeller Synchroner/Asynchroner Empfänger(Receiver)/Sender(Transmitter)). Der vereinfachte interne Aufbau dieses speziell für die Z80-Familie entwickelten USART ist im Blockschaltbild Abb. 33 wiedergegeben; vergleiche hierzu auch unseren Standard-USART 8251 von S.64, der allerdings nur 1 Kanal besitzt.

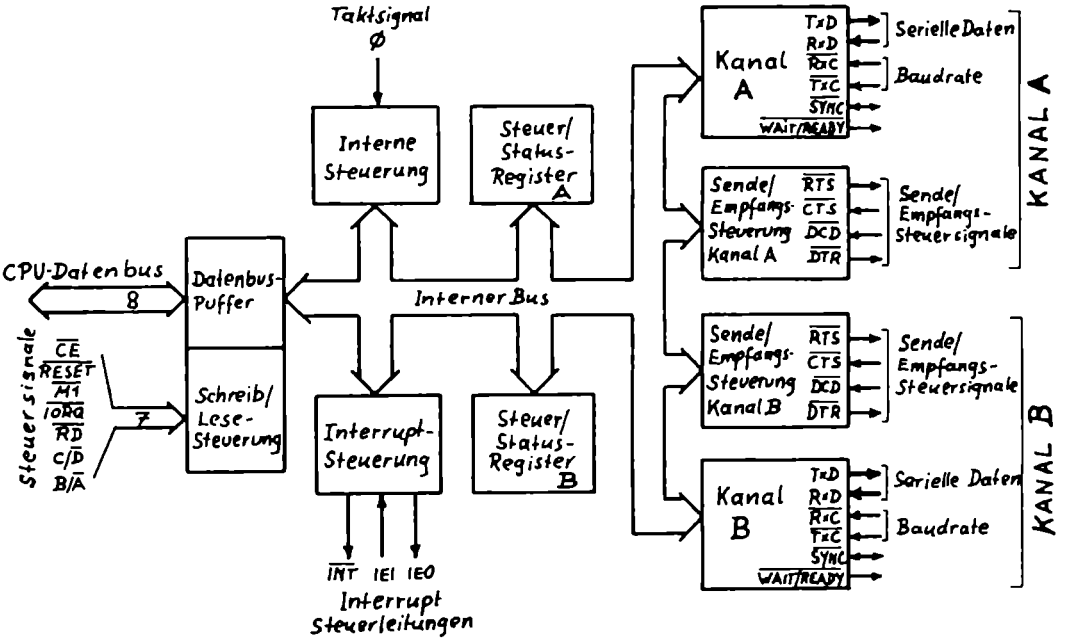


Abb.33: E/A-Baustein Z80-SIO für seriellen Datenaustausch

Die Z80-SIO stellt mit ihren vielfältigen Programmierungsmöglichkeiten eine sehr komplexe Einheit dar, die neben den "Standard-Möglichkeiten" (Wahl der Zahl der Bits pro Zeichen (5-8), der Start- und Stopbits (Rahmung), der Parität usw.) eine fast verwirrende Vielzahl von weiteren programmierbaren Möglichkeiten für den asynchronen und synchronen Datenaustausch und seine Steuerung und Kontrolle bietet. (Näheres siehe Literatur)

Wie alle Peripherie-Bausteine der Z80-Familie, so besitzt auch die Z80-SIO eine integrierte Interrupt-Logik mit zugehörigem Interrupt-Vektor-Register zur Unterstützung des Interrupt-Modus 2 des Mikroprozessors Z80 und zur Bildung einer Interrupt-Prioritätskette (über die Anschlüsse IEI und IEO) mit anderen Bausteinen der Z80-Familie.

Anschluß der Z80-SIO an den Mikroprozessor Z80

Die nachfolgende Abb. 34 zeigt eine typische Schaltung zum Anschluß der Z80-SIO an die Z80-CPU:

- Die gleichlautenden Anschlüsse von CPU und SIO werden miteinander verbunden.
- Über den SIO-Eingang B/$\overline{A}$ erfolgt die Unterscheidung zwischen den beiden Kanälen A und B; im vorliegenden Fall wird diese Unterscheidung durch die Adresse A0 vorgenommen: B/$\overline{A}$=0 bedeutet Kanal A, B/$\overline{A}$=1 bedeutet Kanal B.
- Über den SIO-Eingang C/$\overline{D}$ wird festgelegt, ob Steuersignale (C/$\overline{D}$=1) oder Daten (C/$\overline{D}$=0) an die SIO übermittelt werden sollen; im vorliegenden Fall wird dies durch die Adresse A1 bewirkt.
- Die Adresse A6 liegt am SIO-Freigabeeingang $\overline{CE}$ und bewirkt die lineare Auswahl des Bausteins.

Die beiden seriellen Kanäle A und B besitzen für den Datenaustausch je eine Empfangsleitung RxD und eine Sendeleitung TxD. Die Übertragungsgeschwindigkeit für beide Richtungen muß durch einen Baudrate-Generator festgelegt werden, dessen Taktsignale an $\overline{RxC}$ (Empfangstakt) bzw. $\overline{TxC}$ (Sendetakt) gelegt werden. Weitere Steuerleitungen für die Sende/Empfangssteuerung ("Modemsteuerung") sind für beide Kanäle A und B getrennt verfügbar.

Auch die Z80-SIO besitzt (wie die Z80-PIO) keinen separaten Anschluß für die $\overline{WR}$-Leitung des Mikroprozessors. Das Schreibsignal für die Z80-SIO wird auch hier wieder durch das Fehlen des aktiven $\overline{RD}$-Signales in Zusammenwirken mit den Signalen C/$\overline{D}$ und $\overline{IORQ}$ gebildet.

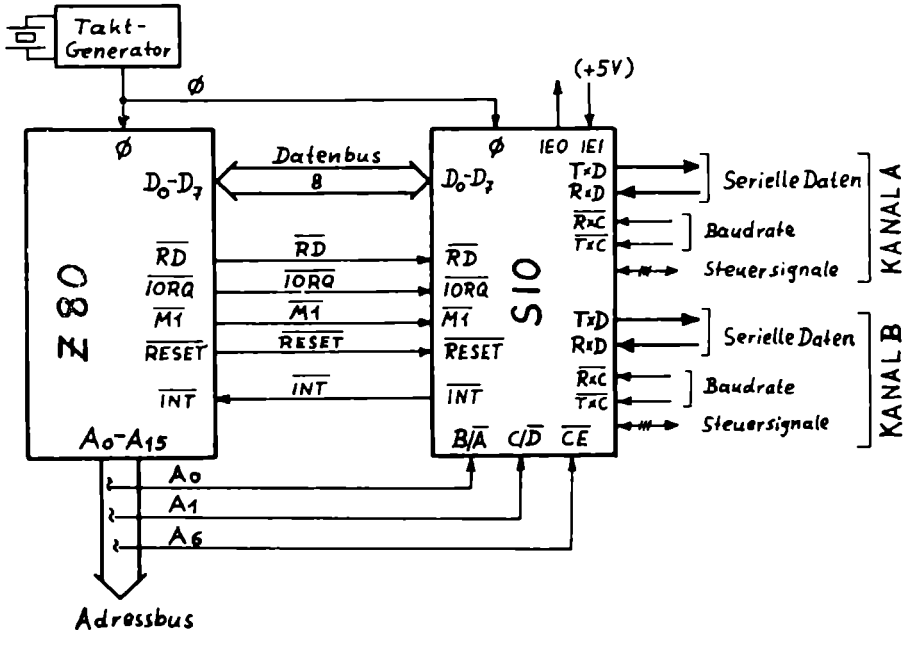


Abb.34: Anschluß der Z80-SIO an den Mikroprozessor Z80

14.33) Interrupt-Prioritätskette bei mehreren E/A-Bausteinen

Treffen bei einem Z80-Mikrocomputersystem, das aus mehreren E/A-Bausteinen besteht, gleichzeitig mehrere Interrupt-Anforderungen von peripheren Geräten ein, so muß eine Priorität der Bedienung der Interrupt-Anforderungen festgelegt werden. Diese Rangfolge wird bei den E/A-Bausteinen der Z80-Familie hardwaremäßig erreicht durch Bildung einer "Prioritäts-Kette" (Daisy Chain). Hierzu besitzen die speziellen Z80-E/A-Bausteine (wie z.B. Z80-PIO und Z80-SIO) je einen Interrupt-Priorisierungs-Eingang IEI (Interrupt Enable In = Interrupt-Freigabe-Eingang) und einen Priorisierungsausgang IEO (Interrupt Enable Out = Interrupt-Freigabe-Ausgang). Die IEI- und IEO-Leitungen der einzelnen E/A-Bausteine werden in der Weise hintereinandergeschaltet, daß jeder nachfolgende Baustein in der Kette eine niedrigere Priorität besitzt als der vorhergehende. Abb. 35 zeigt ein Beispiel für eine solche Prioritätskette mit 4 E/A-Bausteinen:

Der IEI-Eingang des Bausteins 1, der die höchste Priorität erhalten soll, wird an +5V (High) gelegt. Der IEO-Ausgang des Bausteins 1 wird nun mit dem IEI-Eingang des Bausteins 2 mit der 2.-höchsten Priorität verbunden, dessen Ausgang mit dem Eingang des Bausteins 3 usw. Liegt bei einem Baustein der IEI-Eingang auf High-Potential, und liegt keine Interrupt-Anforderung an diesem Baustein vor, so ist automatisch auch der zugehörige Priorisierungs-Ausgang IEO auf High-Potential. D.h., ohne anliegende Interruptanforderungen gilt für die gesamte Kette: IEI=IEO=High.

Ein Interrupt kann von einem Baustein nur angenommen werden, wenn sein IEI-Priorisierungseingang High ist. Die Annahme eines Interrupts (z.B. durch Baustein 3) hat zur Folge, daß an dem betreffenden Baustein IEO=Low wird und damit bei allen nachfolgenden Bausteinen der Kette IEI(=IEO)=Low wird, sie also für Interrupts gesperrt werden. Erst nach Abarbeitung der angeforderten Interrupt-Service-Routine wird durch den Rücksprungbefehl "RETI" der Priorisierungs-

ausgang IEO des "aktiven" Bausteins (3) wieder auf High gelegt und damit für alle nachfolgenden Bausteine der Kette die Interrupt-Empfangsbereitschaft wiederhergestellt.

Auf diese Weise kommt eine Interrupt-Bedienungs-Hierarchie für die einzelnen E/A-Bausteine zustande, wobei auch eine schon laufende Interrupt-Routine durch eine Interrupt-Anforderung höherer Priorität unterbrochen wird und erst später vollends zu Ende ausgeführt wird. (Besitzt ein E/A-Baustein selbst schon mehrere Kanäle - wie z.B. die Z80-PIO -, so ist die Priorität dieser Kanäle intern fest vorgegeben: Kanal A vor Kanal B !)

Die Interrupt-Ausgänge $\overline{INT}$ sämtlicher E/A-Bausteine werden, wie in Abb. 35 gezeigt, miteinander verbunden und führen gemeinsam zu dem Interrupt-Eingang $\overline{INT}$ der Z80-CPU.

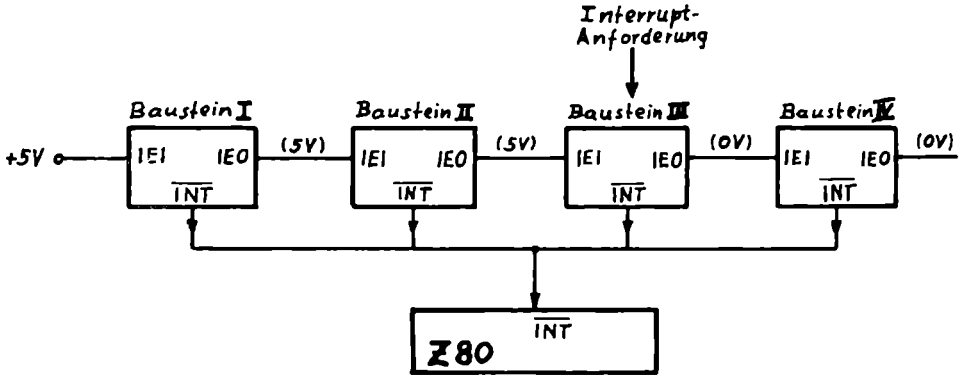


Abb.35: Interrupt-Prioritätskette (Daisy Chain)
aus 4 E/A-Bausteinen der Z80-Familie

14.34) Weitere Peripheriebausteine für das System Z80

Zum Abschluß soll noch auf eine Gruppe von hochintegrierten, programmierbaren Bausteinen hingewiesen werden, die für das Mikrocomputersystem Z80 entwickelt wurden und die den Zentralprozessor von programmintensiven Aufgaben entlasten können.
Zunächst kann an dieser Stelle auf die Seiten 155 und 156 verwiesen werden, wo eine Reihe von programmierbaren Peripheriebausteinen aufgeführt sind, die im System 8080/85 Verwendung finden. Diese allgemein einsetzbaren Bausteine sind zum größten Teil auch für das System Z80 verwendbar.
Daneben werden von den Z80-Herstellern eine Reihe von programmierbaren Peripheriebausteinen angeboten, die auf die speziellen Eigenschaften des Z80-Systems (z.B. die Interrupt-Prioritäts-Kette) zugeschnitten sind. Es sind dies:

- weitere programmierbare Serien-Schnittstellen-Bausteine mit mehreren Kanälen
- programmierbare DMA-Controller für den direkten Speicherzugriff durch Peripheriegeräte (Direct Memory Access)
- universelle programmierbare Zähler/Timer-Bausteine CTC (Counter/Timer Circuit)
- programmierbare Kombinations-Bausteine, in denen verschiedenartige Funktionen für den peripheren Datenaustausch vereinigt sind.

Bezüglich der genauen Beschreibung dieser leistungsfähigen und für den Anwender sehr hilfreichen Bausteine wird auf die umfangreichen Datenbücher der Herstellerfirmen verwiesen (siehe Literaturverzeichnis).

Literaturverzeichnis

SIEMENS: Mikrocomputer-Bausteine Datenbuch 1979/80
Band 1: Mikroprozessor System SAB 8080
Band 2: Speicher
Band 3: Peripherie

SIEMENS: Mikrocomputer-Bausteine Datenbuch 1980/81
Mikroprozessor System SAB 8085

SIEMENS: Makroassembler-Programmiersprache
System SAB 8080/85, 1981

SIEMENS (Benda, H.J. und Hund, F.): Einführung in die Assembler-Programmierung der Mikroprozessoren SAB 8080/85 , 1982

Leventhal, L.A.: 8080A/8085 Programmieren in Assembler, 1980

INTEL: MCS-80/85 Family User's Manual, 1979

INTEL: 8080/8085 Assembly Language Programming Manual, 1979

INTEL: 8080/8085 Floating-Point Arithmetic Library, 1977

INTEL: Components Data Catalog 1983

ZILOG: Z80-CPU Technical Manual, 1977

ZILOG: Z80-PIO Technical Manual, 1977

ZILOG: Z80-SIO Technical Manual, 1977

ZILOG: 1982/83 Data Book

MOSTEK: Z80-Microcomputer Data Book, 1981

MOSTEK: 1982/83 Microelectronic Data Book

Zaks, R.: Programmierung des Z80, 1982

Osborne, A.: Einführung in die Mikrocomputer-Technik,
4. Aufl. 1983

Werkhefte der Universität Tübingen

Reihe A – **Naturwissenschaften**

Nr. 1 Rudolf Schief, Einführung in die Mikroprozessoren und Mikrocomputer. Am Beispiel der Mikroprozessoren 8080/8085/Z80. 6. überarbeitete Auflage
ISBN 3-921552-41-9 DM 19.50

Nr. 2 Mario Dal Cin/Elmar Dilger, Einführung in Microcomputer Pascal.
ISBN 3-921552-14-1 DM 12.25

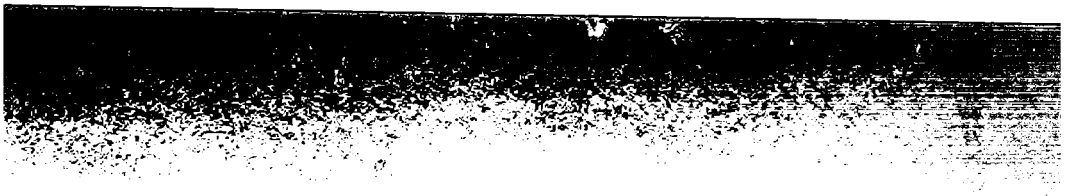
Nr. 3 Manfred Ronge/Heinrich Schmid, Experimentalphysik I. 2. Auflage
ISBN 3-921552-29-X DM 17.25

Nr. 4 Mario Dal Cin/Elmar Dilger (Eds.), Self-Diagnosis and Fault-Tolerance.
ISBN 3-921552-16-8 DM 22.–

Nr. 5 Peter Kramer/Rainer Schultheis (Eds.), Clustering Phenomena in Nuclei.
ISBN 3-921552-19-2 DM 24.–

Nr. 6 Günter Staudt (Hrsg.), Experimentalphysik II. 2. Auflage
ISBN 3-921552-35-4 DM 19.80

Nr. 7 Helmut Eck, Methoden wissenschaftlichen Arbeitens. Eine Einführung für Geographiestudenten.
ISBN 3-921552-33-8 DM 18.60

Nr. 8 Günter Gauglitz Praktische Spektroskopie
ISBN 3-921552-38-9 DM 29.50

ATTEMPTO Verlag Tübingen GmbH
Wilhelmstraße 7
7400 Tübingen 1
Telefon (07071) 292530